Sistemas Informáticos

Curso 2008-2009

Identificación óptica de la posición y orientación de un Vehículo Aéreo no Tripulado

Beatriz Martín Guadaño

Ana Melcón Sanjuán

Daniel Tapia Manganero

Dirigido por:

Dr. José Antonio López Orozco

Dr. José Jaime Ruz Ortiz

Dpto. Arquitectura de Computadores y Automática

Facultad de Informática

Universidad Complutense de Madrid

Resumen

En este documento se describen los aspectos más importantes del Sistema de Localización desarrollado. El objetivo principal de este trabajo consiste en la construcción de un sistema de posicionamiento, por medio de visión estereoscópica, con el que se pueda realizar el seguimiento de un objeto móvil en tiempo real utilizándose para su reconocimiento dos marcas ópticas del mismo color aplicadas al móvil, en entornos cerrados a pequeñas y medias distancias.

Para el correcto uso de las cámaras, previa al posicionamiento, será necesario realizar la calibración de éstas para obtener sus parámetros intrínseco y extrínseco, la distancia focal y la inclinación con respecto al eje Z real.

Por otra parte, los datos calculados por el Sistema serán enviados por UDP a otro sistema que se encargue de tratar esta información para actuar sobre el móvil según sea necesario.

Cualquier objeto móvil puede ser localizado por el sistema, si bien la idea inicial fue emplear un cuatrimotor utilizado por un grupo de investigación de la Facultad de Físicas de la Universidad Complutense de Madrid y controlado por medio del sistema diseñado por otro grupo de trabajo de la asignatura de Sistemas Informáticos de la Facultad de Informática.

En este documento se muestran, además, diferentes ejemplos de localización y seguimiento de un móvil llevados a cabo con el sistema.

Palabras clave

Visión estereoscópica, tiempo real, posicionamiento, calibración, UDP

Abstract

This document describes the most important aspects of the Location System which has been developed. The main aim of this work consists of the development of a positioning system, by means of stereoscopic imaging, with which it can be carried out the tracking of a non-static object in real-time, using for its recognition two optical signs of the same colour applied to the object, within an enclosed space for short and medium distances.

For the purpose of making a correct use of the cameras, previously to positioning tasks, it is required to calibrate them in order to obtain its intrinsic and extrinsic parameters, its focal length and its inclination with respect to the real axis Z.

In addition, the data calculated by the system are sent through UDP to another system in charge of processing this information and act on the object as needed.

Even though any non-static object can be located by the system, the initial idea was making use of a four-engined aeroplane used by an investigation group from the Physics Faculty of the "Universidad Complutense de Madrid" and controlled through a system designed by another working group of the "Sistemas Informáticos" subject from the Computer Science Faculty.

Finally, this document also shows different examples of location and tracking of the mobile carried out by the system.

Keywords

Stereoscopic imaging, real-time, positioning, calibration, UDP

Autorización

Autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Beatriz Martín Guadaño

Ana Melcón Sanjuán

Daniel Tapia Manganero

Contenido

Índice de ilustraciones

Capítulo 1. Introducción

1.1. Objetivos

El objetivo inicial del proyecto era diseñar un sistema de visión para el posicionamiento de un cuatrimotor. La estima de la posición y orientación del cuatrimotor era esencial si se deseaba poder controlar éste mediante un PC. Esto llevó a una serie de requisitos imprescindibles para el sistema:

- Debía ser rápido, ya que la dinámica del cuatrimotor es muy alta. Esto obligó a buscar algoritmos sencillos y una configuración de las cámaras que no llevara a un excesivo tiempo en el cálculo de la posición.
- Preciso, en el sentido de que no debía cometer errores en la estima de la posición, puesto que provocaría un control erróneo del cuatrimotor.
- Fiable y robusto, era necesario que en caso de pérdida de la imagen o errores por reflejos y otros elementos espúreos en la imagen no se obtuviera una posición del cuatrimotor que no correspondiera. Esto era esencial si queríamos que el control del cuatrimotor no se desestabilizase.
- Que garantizase unos requisitos de tiempo mínimos, puesto que se utilizaba para controlar un sistema en tiempo real.

Con esto requisitos iniciamos el proyecto pero, puesto que con un pequeño esfuerzo sería fácil convertir el sistema de posicionamiento del cuatrimotor a un sistema de posicionamiento para el laboratorio, se han generalizado algunas de las especificaciones para implementar un sistema de visión para posicionamiento de sistemas en tiempo real. De este modo, nos hemos abstraído de la idea inicial del cuatrimotor en la construcción del sistema de posicionamiento, pero dotándole de las características necesarias para poder realizar el control de posición del cuatrimotor, puesto que era el requisito inicial.

Por lo tanto, el objetivo del presente proyecto es la construcción de un sistema basado en visión estereoscópica para el posicionamiento de móviles en el laboratorio.

Los requisitos del sistema para que pueda adaptarse a diferentes configuraciones son, además de los indicados anteriormente (rápido, preciso, fiable, robusto y que garantice unas restricciones de tiempo mínimas), que sea totalmente configurable, lo cual es fundamental para adaptarse a diferentes entornos, y fácilmente reprogramable y adaptable a diferentes cámaras.

En el Capítulo 6 se presentan distintos experimentos para mostrar el amplio abanico de posibilidades de posicionamiento (para control de sistemas) en el laboratorio.

1.2. Estado del arte

En este apartado se expondrán, en primer lugar, diversos trabajos relacionados con el proyecto desarrollado, y más adelante distintos sistemas de localización comparándolos con el usado en nuestro proyecto.

1.2.1. Trabajos relacionados

En los últimos años ha sido cada vez mayor el número de proyectos que emplean la visión como medio de comunicación con el exterior, haciendo a los robots más independientes y autónomos. De esta forma, gracias al uso de una o más cámaras, el robot es capaz de conocer el entorno que le rodea o su localización en él.

Hay que tener en cuenta que todos los métodos de visión se basan en la captura de imágenes que, posteriormente, son procesadas para extraer de ellas la información necesaria.

El número de cámaras empleadas depende de las necesidades del sistema, algunos sistemas cuentan con una sola cámara, otros con dos, llevando a cabo lo que se conoce como *visión estereoscópica,* y otros con más de dos. Todos estos sistemas, además, pueden complementarse con el uso de algún sensor de distinta naturaleza.

A continuación se van a mencionar algunos ejemplos de proyectos que emplean la visión con distinto número de cámaras.

Para empezar, se expondrán dos ejemplos de visión monocular, es decir, que usan una sola cámara para la obtención de la información del entorno necesaria. Con el uso de una sola cámara no se obtiene información 3D de la escena, por ello hay que combinar la información obtenida de la imagen con un procesamiento más a fondo de ella, o bien con algún otro sensor que aporte nuevos datos al sistema.

En el caso de un robot jugador de Ping-Pong [J. J. Rodrigo, L. Acosta, J.A. Méndez, S. Torres], por ejemplo, su sistema de visión está formado por una única cámara y éste se basa en la detección de la pelota y de la sombra que la misma proyecta sobre la mesa. Las tareas del sistema de visión son las de procesar la imagen y estimar la trayectoria que seguirá la pelota en el espacio. Su cuerpo, además, consta de dos articulaciones que permiten llevar a cabo las tareas del sistema de control, interceptar la bola y orientarla en el espacio, y golpearla según la estrategia de juego.

Por otra parte, y como ejemplo de un sistema que emplea una cámara combinada con otro sensor, se puede mencionar un sistema de medida de visión a distancia para vehículos submarinos no tripulados [Muljowidodo K., Mochammad. A. Rasyid.,

Sapto Adi N., and Agus Budiyono] que emplea una cámara y un puntero láser. Este sistema predice la distancia horizontal entre el vehículo y la pared o la distancia vertical entre el vehículo y la superficie por debajo de él. Por medio del procesamiento de la imagen, detecta el láser y calcula la distancia basada en la posición del láser en la imagen.

Otros sistemas, como se ha explicado anteriormente, emplean dos cámaras para la obtención de la información del entorno, de forma que con ambas se pueden obtener, mediante el procesamiento de las imágenes, las coordenadas 3D buscadas.

Un ejemplo de estos sistemas de visión estereoscópica es un robot de gran precisión diseñado para la desactivación de bombas [Hengnian Qi, Wei Wang, Liangzhong Jiang, Luqiao Fan, 2007]. Tras la localización de la bomba, el manipulador será controlado para llegar a ella una vez conocidas sus coordenadas. El sistema del robot está formado por dos partes, el subsistema de visión estereoscópica y el subsistema de control de movimiento.

AUTOPÍA [Proyecto AUTOPÍA, CSIC], [Vicente Milanés, Enrique Onieva, Teresa de Pedro, Ricardo García, Javier Alonso, Joshué Pérez, Carlos González], otro proyecto ejemplo de estos sistemas, pretende transferir las técnicas desarrolladas para el control de robots autónomos al control de vehículos. Clavileño es un modelo de Citroën C3 diseñado por el CSIC junto a varias universidades y con la colaboración de Citroën, al que se ha dotado de dos cámaras para que, mediante visión estereoscópica, pueda detectar y esquivar obstáculos. El computador embarcado en el coche utiliza una estrategia de control basada en la lógica borrosa que permite simular el comportamiento de un chófer humano con el objetivo de encontrar un conjunto de estrategias que permitan construir un conductor automático.

Figura 1: Proyecto AUTOPÍA

Por otra parte, también existen proyectos que emplean más de dos cámaras para la visión, como es el caso de un sistema para la medición automática y monitorización de fuegos [J. R. Martínez-de Dios, A. Ollero, L. Merino, y B. C. Arrue, 2006] mediante cámaras visuales y de infrarrojos desarrollado por el Grupo de Robótica, Visión y Control de la Universidad de Sevilla. Este sistema es capaz de medir automáticamente y en tiempo real parámetros del frente de llamas tales como la posición, la velocidad y altura de llamas empleando cámaras visuales y de infrarrojos con diferentes vistas en localizaciones fijas o en medios terrestres o aéreos.

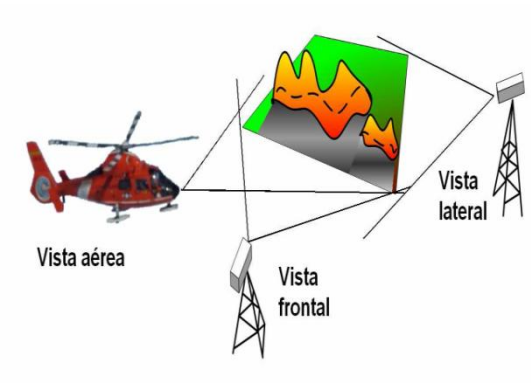


Figura 2: Despliegue de estaciones de adquisición en los experimentos

Figura 3: Estación frontal en unos experimentos de mayo de 2006

Como se ha podido observar, son numerosos los sistemas que emplean la visión para el reconocimiento del medio.

En algunos casos los sistemas evitan la colisión de un móvil con algún obstáculo midiendo las distancias del móvil con los objetos cercanos (submarinos no tripulados y AUTOPÍA).

En otros casos, los sistemas se encargan de localizar un objeto en el entorno para conocer su posición (jugador de “Ping-Pong”, bombas, medición de fuegos) y poder ejecutar las acciones necesarias según el proyecto de que se trate (golpear la pelota de Ping-Pong, desactivar bombas o monitorizar el fuego).

Nuestro sistema se enmarca dentro del segundo grupo, es decir, el sistema se encarga de la localización de un móvil. Como se ha explicado anteriormente, la idea inicial se basó en un cuatrimotor que, tras conocer su posición y orientación, podrá ser controlado automáticamente desde un computador.

1.2.2. Otros sistemas de localización

A continuación se realiza un repaso por los distintos sistemas sensoriales [A. Gardel Vicente, 2004], existentes en la actualidad, que permiten la localización de un objeto en el entorno, analizando sus características.

Los robots cuentan en su arquitectura básica con sensores. Estos sensores, a pesar de ser de diferentes tipos y naturaleza, son complementarios, de hecho, la fusión de información proveniente de distintos sistemas sensoriales ha sufrido un considerable auge.

Una posible clasificación de los sensores está basada en la forma de controlar el robot, de manera interna o externa.

- Sensores internos: controlan accionamientos de la estructura mecánica del robot y permiten implementar técnicas de posicionamiento relativo como los potenciómetros o los sensores odométricos e inerciales.

- Sensores externos: detectan la posición del robot respecto al entorno. Estos sensores pueden ser de contacto (antenas, bumpers, dedo robot, array sensor, etc.) o de no contacto como los ultrasonidos, la visión, los infrarrojos o los sensores radio.

Otra posible clasificación de los sensores se basa en el posicionamiento llevado a cabo, es decir, relativo o absoluto.

- Sensores relativos: miden la posición del robot a partir de su posición anterior, por lo que es necesario conocer su posición inicial, pero presentan el problema de que, con el tiempo, acumulan errores de cálculo, haciendo necesaria una nueva localización externa. Estos sensores son los odométricos y los inerciales, de los que algunos ejemplos son los potenciómetros, encoders, giróscopos y acelerómetros.

- Sensores absolutos: miden la posición del robot respecto un sistema de referencia externo, independientemente de su posición anterior. Estas técnicas son usadas por sensores externos. Los sensores explicados a continuación son sensores absolutos.

En tareas de posicionamiento a corta distancia donde es necesaria una localización muy precisa, los sensores empleados son los mecánicos y los magnéticos. Los sensores mecánicos no tienen problemas de interferencias y sus errores están sujetos a las incertidumbres de las medidas de las piezas y holguras en los movimientos mecánicos. Mientras que los sensores magnéticos son más baratos que los mecánicos y no tienen que estar en contacto con el objeto pero sí muy próximos a él.

Dentro de los posibles sensores ópticos, la luz infrarroja es utilizada frecuentemente para tener menos interferencias con otras fuentes luminosas, pasando inadvertida para los humanos. Los sensores ópticos tienen una elevada precisión y rápida respuesta aunque están limitados por el alcance de la fuente de luz. El precio de estos sensores es más elevado que el de otros.

Dentro de los sistemas de visión artificial se puede hacer uso de marcas artificiales y/o marcas naturales. La ventaja de las artificiales con respecto a las naturales es que son más sencillas de reconocer en el entorno porque su apariencia no depende del entorno, hecho que hace a éstas más robustas aunque, por el hecho de existir ya en el entorno, las naturales no hay que incorporarlas.

Los dispositivos de ultrasonidos [Diego Pérez de Diego, 2006] utilizan frecuencias por encima de los 20KHz, imperceptibles por el oído humano. Son baratos, compactos y ligeros, pero su precisión no es muy elevada y depende de las condiciones del medio en el que se encuentren.

El funcionamiento básico de los ultrasonidos se muestra en la siguiente figura, donde se tiene un receptor que emite vibraciones periódicas en el tiempo y en el espacio a una velocidad de transmisión de 340m/seg. Estas vibraciones rebotan sobre un determinado objeto y la reflexión de esas vibraciones es detectada por un receptor de ultrasonidos. Midiendo el tiempo que transcurre entre la emisión del sonido y la percepción del eco se puede establecer la distancia a la que se encuentra el objeto que ha producido la reflexión de la onda sonora.

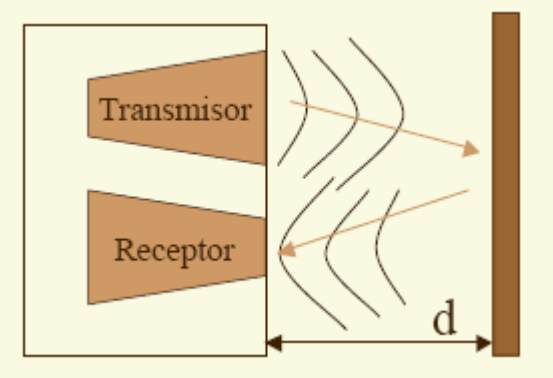


Figura 4: Funcionamiento básico de los ultrasonidos

Los sistemas radio, GSM, etc. se utilizan para la localización y posicionamiento robusto de vehículos y móviles, pero tienen una precisión baja y son susceptibles de interferencias por otras ondas de radio.

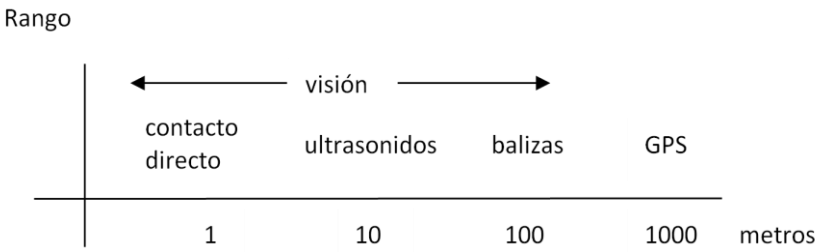


Figura 5: Rango de distancias de aplicación de los sistemas de posicionamiento

Como se puede ver en la figura anterior, según para qué distancias se trate, son mejores algunos sistemas u otros.

A distancias cortas, los sistemas más útiles son los de contacto directo (sensores mecánicos y magnéticos) ya que realizan los cálculos con gran precisión.

Para distancias alrededor de los 10m los dispositivos más adecuados son los ultrasonidos. Este sensor espera un tiempo hasta que las vibraciones emitidas desaparezcan y esté preparado para recibir el eco producido por el objeto, por lo que existe una distancia mínima a partir de la cual el sensor mide con precisión. A distancias más pequeñas, el sensor, por tanto, no es preciso.

Sin embargo, las balizas permiten posicionar un objeto a 100m de distancia mediante la comparación de señales enviadas por las fuentes emisoras.

A distancias de 1000m, el sistema más adecuado es el GPS, Global Positioning System, o Sistema de posicionamiento y navegación basado en satélites.

En el rango de distancias de 0 a 100m, se puede emplear la visión para el posicionamiento de un objeto. En el apartado anterior “Trabajos relacionados” se pueden comprobar las distintas utilidades de los sistemas de visión y las distintas precisiones conseguidas en ellos. Por ejemplo, el sistema desarrollado para la desactivación de bombas [Hengnian Qi, Wei Wang, Liangzhong Jiang, Luqiao Fan, 2007], necesita unos cálculos muy precisos y a muy pequeñas distancias, frente al de monitorización de incendios [J. R. Martínez-de Dios, A. Ollero, L. Merino, y B. C. Arrue, 2006] en el que la precisión es importante, pero no se consideran distancias tan cortas y por tanto, el error en los cálculos es admisible.

Tras este análisis de los dispositivos sensoriales existentes, cabe destacar que nuestro sistema lleva a cabo el posicionamiento de manera absoluta, puesto que no es necesario en un momento dado conocer la posición anterior del robot, y externo al robot, pudiendo además emplear cualquier robot o móvil para su localización.

El sistema emplea la visión como medio de comunicación con el entorno puesto que, por medio de las capturas de dos cámaras, se obtiene la información necesaria para el cálculo de la posición y de la orientación.

También hay que indicar que las marcas empleadas para la visión son artificiales, pudiendo emplear también marcas naturales, siempre y cuando sean bien distinguibles en el entorno.

En nuestro caso, con las webcam de que disponemos y las pruebas realizadas con el sistema, la distancia de aplicación para el posicionamiento y la precisión dependen directamente del tamaño del objeto de referencia.

Si el objeto de referencia es pequeño, la precisión será mayor cuanto más cerca esté el objeto de las cámaras; o lo que es lo mismo, a mayor distancia, más posibilidades hay de que las cámaras no vean el objeto o se confunda con el entorno y, por lo tanto, menos fiabilidad habrá en el cálculo de las medidas.

Sin embargo, si el objeto de referencia es más grande, se puede calcular la posición a mayor distancia, siempre y cuando el objeto esté en el campo de visión de las cámaras.

Capítulo 2. Diseño Hardware

2.1. Introducción

Antes de comenzar el desarrollo de la aplicación se necesitaba especificar qué tipo de cámaras se utilizarían para la captura de las imágenes. Las opciones posibles eran usar dos webcams con una buena calidad de captura o cámaras de mayor calidad pero que utilizan su propio hardware para su conexión con el computador. Tras un análisis se decidió utilizar las webcams, puesto que la conexión con el computador y, por tanto, con la aplicación desarrollada sería mucho más sencillo, ya que se conecta al computador mediante una interfaz USB y no es necesario configurar el software ni los drivers.

Las Webcams utilizadas durante el desarrollo de la aplicación fueron del modelo Logitech Quickcam Pro for Notebooks de 1,3 MP, puesto que ya se habían utilizado para un proyecto fin de carrera anterior relacionado con visión y proporcionaron buenos resultados.

Antes de decantarnos definitivamente por las webcams era necesario realizar unas pruebas para determinar la velocidad de captura real, puesto que para el sistema de medida lo más prioritario es la eficiencia y la rapidez del envío de los datos, ya que de esto depende el correcto control del cuatrimotor. Una vez realizadas estas pruebas se determinó que el uso de las webcams era viable, ya que la tasa de captura medida fue cercana a 15 fps.

Para la localización del objeto era necesario definir un sistema mediante el cual las tareas tanto de posicionamiento como de cálculo de la orientación fueran lo más sencillas posibles.

Como era necesario el cálculo de la orientación era indispensable contar con, al menos, dos objetos distinguibles para poder realizar los cálculos necesarios. Se decidió el uso de dos objetos del mismo color para así, teniendo dos puntos de referencia, poder calcular el vector de la orientación del aparato y su correspondiente ángulo respecto a los ejes de coordenadas.

Si se necesitara calcular la orientación real del aparato, los dos objetos de referencia deberían ser de dos colores diferentes, pero esta opción se desestimó ya que esta funcionalidad conlleva la realización del procesamiento de las imágenes por duplicado, y en esta primera fase del proyecto era más importante que el sistema fuera lo más rápido posible.

Además, el objetivo inicial era utilizar el sistema de visión como sensor de posicionamiento para realizar un control de posición (entorno a una posición de equilibrio), por lo que no habría grandes variaciones del ángulo medido.

Para los objetos de referencia se hicieron pruebas con distintos colores, llegando a la conclusión de que los colores más apropiados eran el rojo y el naranja, puesto que eran los que menos problemas daban en un entorno controlado, ya que son muy distinguibles. Otros colores con los que se realizaron pruebas fueron el verde y el azul, dando bastantes problemas al confundirse con el entorno después del procesado de las imágenes, por lo que se desestimaron.

Para la completa utilización del sistema son necesarias dos webcams que se encargan de capturar las imágenes del cuatrimotor. Estas webcams estarán conectadas a un ordenador donde se ejecuta la aplicación, y que estará conectado a otro ordenador de su misma red, al que enviará los datos calculados a través de UDP para que este los procese y realice el control del cuatrimotor. El esquema de esta arquitectura se representa en la siguiente figura.

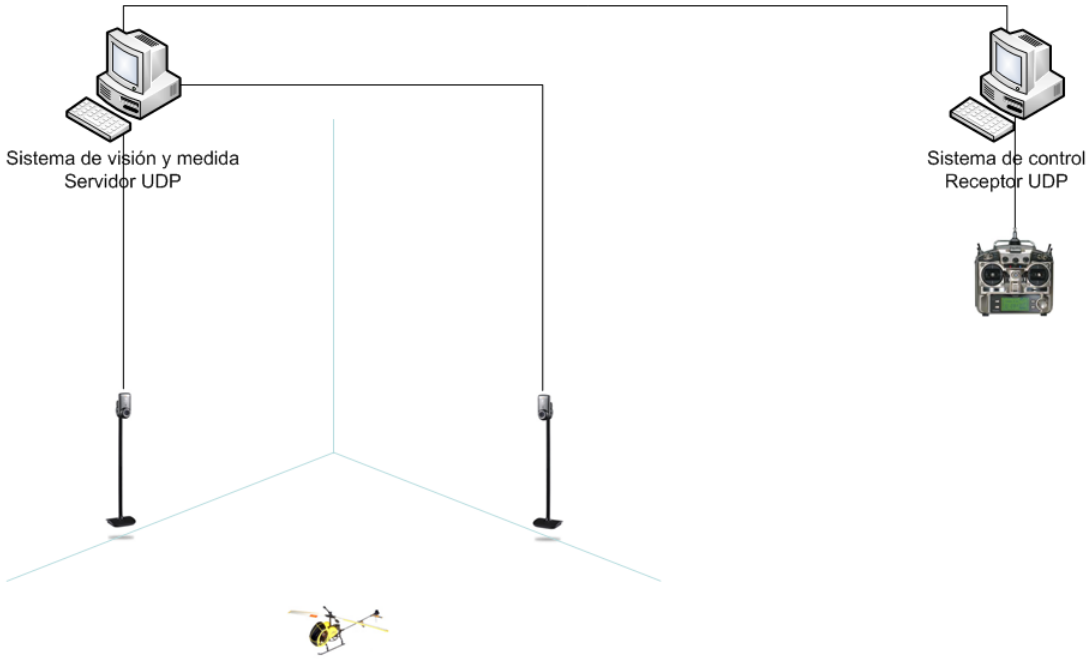


Figura 6: Diagrama de la arquitectura

2.2. Cámaras

2.2.1. Modelo pin-hole

El modelo de cámara pin-hole [Josep Isern González, 2003], [A. Gardel Vicente, 2004], [Javier García Ocón, 2007] está basado en la geometría proyectiva: la proyección de un punto de la escena se obtiene de la intersección de una recta que pasa por este punto y el foco con el plano imagen.

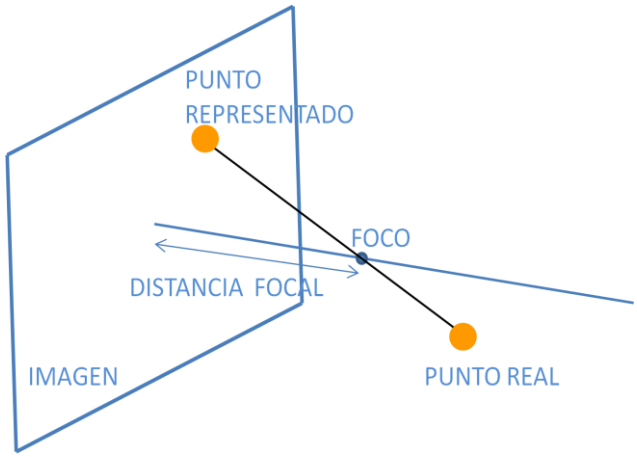


Figura 7: Esquema de una cámara pin-hole

El modelo pin-hole consiste en un centro óptico en donde convergen todos los rayos de la proyección, y un plano de imagen en el cual la imagen es proyectada. El plano de imagen está ubicado a una distancia igual a la distancia focal del foco y perpendicular al eje óptico.

El modelo pin-hole describe la proyección de un objeto real de la escena en un objeto representado de la imagen. Para modelar la proyección es necesario realizar varias transformaciones y referirse a varios sistemas de coordenadas distintos. Estos son los sistemas que aparecen en el modelo de proyección:

-*Sistema de coordenadas del mundo*: son las coordenadas que describen la posición del punto 3D respecto de la escena, las coordenadas del mundo real.

-*Sistema de coordenadas de la cámara*: son las coordenadas que describen la posición del punto 3D respecto de la cámara, las coordenadas locales a la cámara.

-*Sistema de coordenadas de la imagen*: son las coordenadas que describen la posición del punto 2D respecto del plano de la imagen.

Básicamente, este modelo aplica una matriz de proyección para transformar las coordenadas 3D de los puntos del objeto (coordenadas del mundo) en coordenadas 2D de la imagen:

Matriz de transformación x Punto real = Punto imagen

Siendo:

-Matriz de transformación: la matriz que permite convertir unas coordenadas de los ejes de coordenadas locales de una cámara a coordenadas de los ejes del mundo real.

-Punto real: punto está expresado en coordenadas del mundo real.

-Punto imagen: punto expresado en coordenadas 3D respecto a los ejes locales de la cámara. Se obtienen estas coordenadas 3D, pasando las coordenadas obtenidas de la imagen expresadas en píxeles a coordenadas 2D y añadiendo la coordenada que falta para completar las 3D.

2.2.2. Disposición de las cámaras

En este apartado se expone el sistema de medida, la situación de cada una de las cámaras y el espacio de medida disponible.

Para el sistema de medida desarrollado se dispone de dos cámaras colocadas en planos perpendiculares entre sí. De esta manera podemos obtener las coordenadas en tres dimensiones utilizando los valores que obtenemos de cada cámara, que son en dos dimensiones. Esto se explicará más detalladamente en el siguiente capítulo.

A través de este sistema los cálculos son más sencillos que si las cámaras estuvieran colocadas en el mismo plano, por lo que los valores calculados se obtendrán más rápidamente, algo que es imprescindible dadas las necesidades de velocidad y eficiencia requeridas para el control del objeto que se esté utilizando.

En la siguiente figura se expone un esquema del sistema de medida, la situación de las cámaras y el espacio de medida de que se dispone:

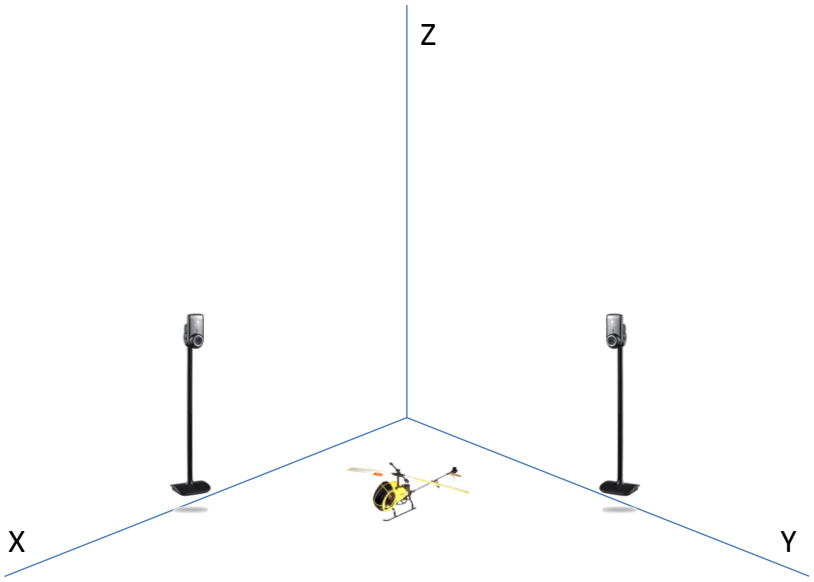


Figura 8: Disposición de las cámaras en el espacio

Las cámaras pueden estar situadas en cualquier punto del plano en el que se encuentran, inclinadas o no, de la siguiente manera:

La cámaraXZ es la cámara situada en el plano XZ. Puede estar posicionada en cualquier punto de ese plano desplazada a través de los ejes X y Z del origen de coordenadas del mundo real una distancia siempre positiva. Además puede tener una cierta inclinación con respecto al eje Z, siendo positivo el ángulo si la cámara está inclinada hacia el cuadrante con X e Y positivas, y negativo si está inclinada hacia el cuadrante con X positiva e Y negativa.

La cámaraYZ es la cámara situada en el plano YZ. Como la primera, ésta puede estar desplazada del origen de coordenadas a través de los ejes Y y Z una distancia positiva. La cámara YZ también puede estar inclinada con respecto al eje Z, de manera que el ángulo de inclinación será positivo si la cámara está inclinada hacia el cuadrante con X e Y positivas, y será negativo si está inclinada hacia el cuadrante con X negativa e Y positiva.

En las siguientes figuras se puede observar la disposición de las cámaras en el mundo real y sus ejes, en primer lugar sin inclinación, y después teniendo ambas cierta inclinación.

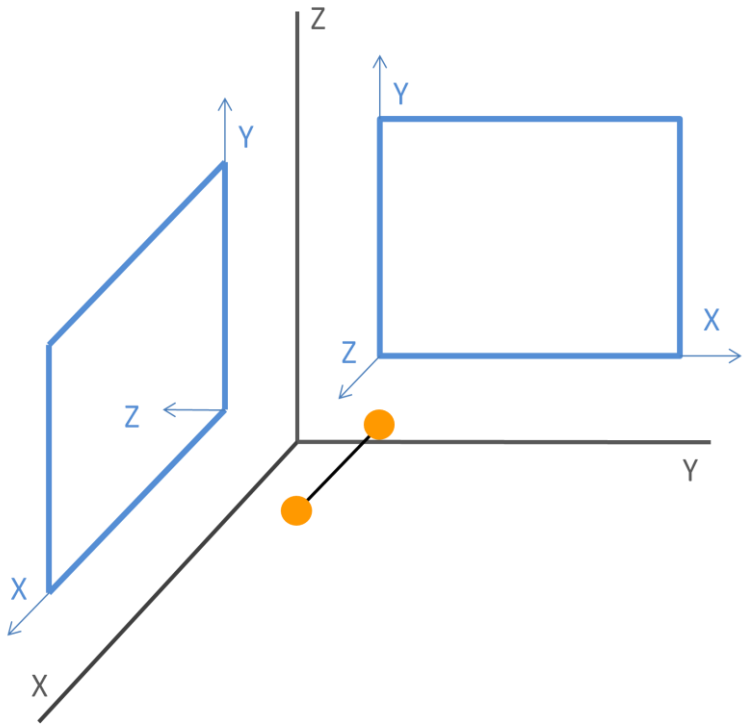


Figura 9: Disposición de las cámaras sin inclinación

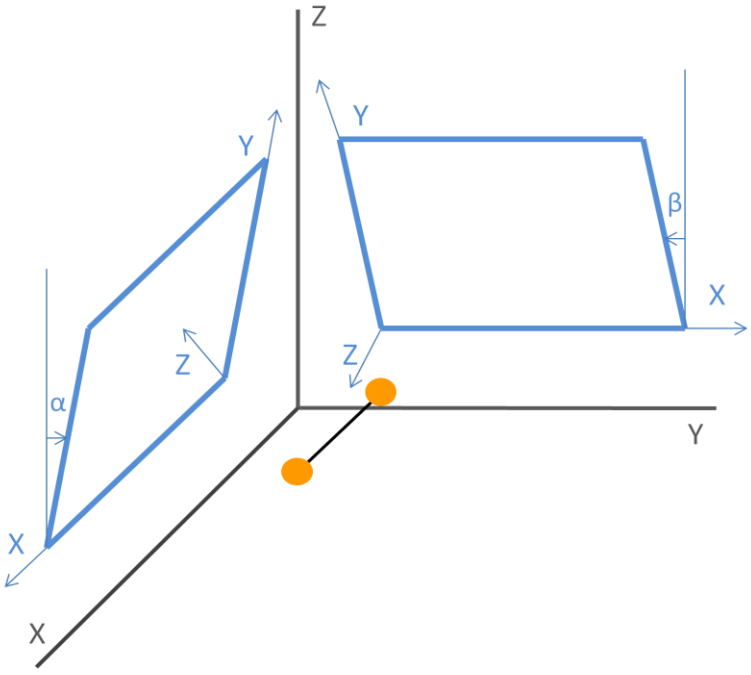


Figura 10: Disposición de las cámaras inclinadas

2.2.3. Calibración y parámetros de las cámaras

La calibración de una cámara es el proceso que permite estimar los valores de los parámetros intrínsecos y extrínsecos que definen las condiciones de formación de la imagen dentro del campo de la visión. Es, por tanto, un procedimiento que trata de

conocer cómo una cámara proyecta un objeto 3D en el plano de la imagen para así poder extraer información métrica a partir de las imágenes.

2.2.3.1. Métodos de calibración

En este apartado se expondrá la clasificación de los métodos de calibración [Josep Isern González, 2003], [Javier García Ocón, 2007], explicando más detalladamente algunos de ellos.

Los métodos de calibración "clásicos", o calibración fotogramétrica, sitúan una serie de puntos 3D cuyas posiciones respecto al sistema de coordenadas del mundo real son conocidas con un buen nivel de precisión.

Por otra parte, los métodos de autocalibración se caracterizan por no necesitar conocimiento de ningún dato de la escena. Éstos se basan en el movimiento de una cámara observando una escena estática, a partir de desplazamientos y usando únicamente la información de la imagen, agregando progresivamente correspondencias entre puntos de diferentes imágenes.

Calibración fotogramétrica

La calibración fotogramétrica consiste en el cálculo de los parámetros intrínsecos y extrínsecos de la cámara a partir de un conjunto de puntos de control, conocidas las coordenadas 3D de esos puntos y midiendo las correspondientes coordenadas 2D en una imagen obtenida con dicha cámara.

Los métodos de calibración fotogramétrica se pueden clasificar según la obtención de los parámetros de la siguiente manera:

Calibración directa de parámetros o calibración explícita: se obtienen los valores de los parámetros intrínsecos y extrínsecos directamente. Ejemplos de estos métodos son [Batista et al., 1998], [Heikkilä and Silvén, 1996] y [Tsai, 1987].

Recuperación desde la homografía o matriz de proyección o calibración implícita: se calculan los valores de los parámetros a partir de los valores de la homografía o matriz de proyección. De algunos de estos parámetros no se puede conocer el valor exacto. Ejemplos de estos métodos son [Ahmed et al., 1999], [Faugeras, 1993] y [Zhang, 1998].

Autocalibración

Los métodos de calibración clásicos no permiten que se pueda realizar una calibración cuando se está utilizando la cámara en tareas visuales. Si la cámara realiza algún tipo de movimiento, sus parámetros extrínsecos variarán. Además, la configuración interna de una cámara puede variar durante la ejecución de tareas.

Los métodos de autocalibración no cuentan con puntos de referencia en los que se conoce la situación exacta de cada uno de ellos. En su lugar, se utiliza la correspondencia de puntos detectados en la escena a lo largo de una secuencia de imágenes. Esto permite que un sistema que lleve incorporada una cámara pueda realizar de forma automática la calibración de ésta en cualquier momento, ya que simplemente necesita adquirir una secuencia de imágenes de su entorno.

Una posible clasificación de estos métodos de calibración es según la variación de los parámetros intrínsecos:

Fijos: se considera que la configuración de la cámara permanece estable entre las diferentes imágenes captadas. En caso de que haya alguna variación, es necesario repetir el proceso de calibración. Algunos ejemplos de estos métodos son: [Maybank y Faugeras, 1992], [Luong et al., 1992] y [Lourakis and Deriche, 1999].

Variables: se estudia el efecto de la variación del zoom. Algunos métodos consideran que esta variación afecta sólo a la distancia focal mientras que otros suponen que también afecta al centro del eje óptico de la cámara. Ejemplos de estos métodos son: [Willson, 1994], [Pollefeys et al., 1999] y [Hartley et al., 1999].

2.2.3.2. Parámetros de las cámaras

Parámetros intrínsecos

Los parámetros intrínsecos son aquellos que definen la geometría interna y la óptica de la cámara. Son constantes en tanto no varíen las características y posiciones relativas entre la óptica y el sensor imagen.

Distancia focal

La distancia focal de una cámara es la distancia existente entre ésta y su foco. Puesto que la lente de la cámara es convergente, la distancia focal es positiva.

En la Figura 11 se puede ver el esquema de la cámara, en la que el foco se encuentra a una distancia del plano de la imagen igual a la distancia focal. Además, como se verá en el Capítulo 3, debido a la lente convergente de la cámara, la imagen es invertida.

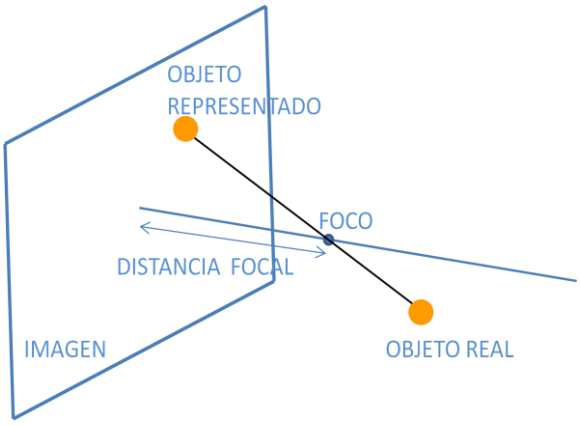


Figura 11: Esquema de la distancia focal

Para el cálculo de la distancia focal de una cámara se van a utilizar dos objetos de referencia y sus representaciones en la imagen de la cámara como se puede observar en la siguiente figura.

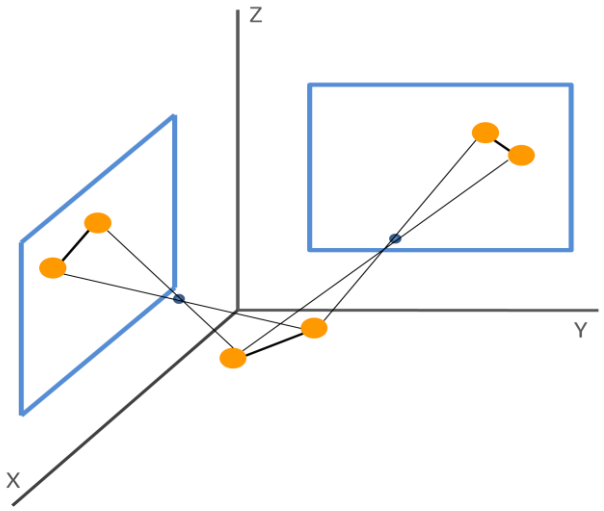


Figura 12: Calibración de la distancia focal

El foco de una cámara es el punto donde convergen los rayos de luz originados desde un punto en el objeto observado, por lo tanto trazando una recta desde el objeto y que pase por el foco de la cámara, en la intersección de la recta con el plano de la cámara se obtiene la representación del objeto.

O como en este caso, en el proceso inverso, todas las rectas trazadas entre los objetos del mundo real y los objetos representados en la imagen pasarán por el foco de la cámara.

Por ello, se conocerá el foco como la intersección de dos o más rectas que vayan desde un objeto hasta su representación en la cámara.

Para empezar y puesto que se emplean dos objetos de referencia, se calcula, para cada objeto, la recta que une el objeto en su posición real con su representación en la cámara. Estas rectas pasarán ambas por el foco, así que será en la intersección de ambas donde se obtendrá éste.

La intersección de estas rectas, debido a imprecisiones en la estima de algún parámetro de la cámara, es probable que no sea exacta, por lo que se calculará el punto situado a la menor distancia posible en dos rectas que se cruzan (véase en el Capítulo de Localización, cómo se resuelve el sistema de ecuaciones que se plantea para el cruce de líneas).

El punto obtenido es el foco estimado de la cámara, así que para calcular la distancia focal de dicha cámara basta con calcular la distancia desde el foco hasta el centro de la cámara. Ésta será la distancia focal estimada puesto que, por los errores anteriormente explicados, puede que no sea exacta.

Tras hallar el foco de la cámara, se calcula el error en este cálculo de forma que el usuario puede conocer la certeza de este valor calibrado.

Este error depende de la distancia entre las rectas que se cruzan, la desviación del foco estimado con respecto al eje óptico y la distancia entre el foco estimado y el punto del eje óptico que se encuentra a una distancia igual a la focal de la cámara.

Factor de conversión pixel-milímetros

Las coordenadas 2D obtenidas están expresadas en forma de píxeles, así que se ha hallado el número de píxeles por milímetro que usan las cámaras para transformarlas a medidas del Sistema Internacional de Unidades (SI).

Con dos objetos de referencia colocados en un plano paralelo al plano de la cámara se han obtenido resultados de las medidas conseguidas en las imágenes comparándolas con las medidas reales. El sistema empleado queda así:

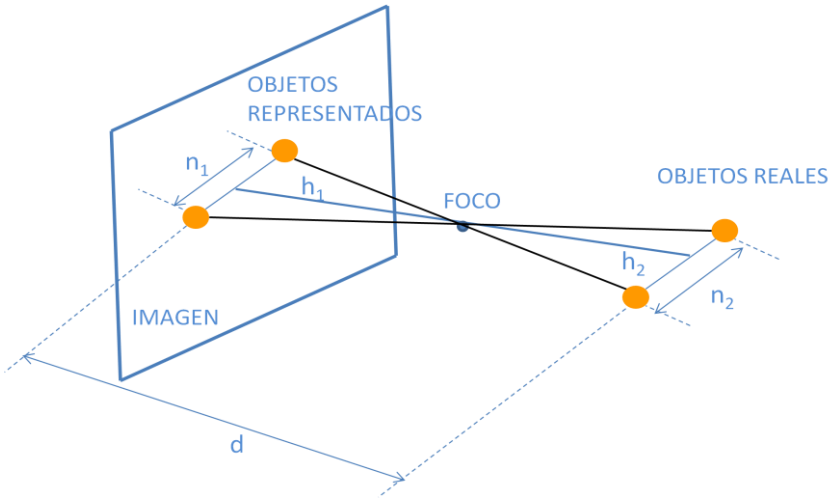


Figura 13: Sistema empleado para el cálculo del valor del pixel

En este caso se busca el valor n_1, puesto que será la relación n_1/n_2 la que permita conocer el tamaño del pixel.

Los datos conocidos son:

-d: distancia entre el plano de la cámara y el plano de la imagen.

-n_2: distancia real entre los objetos de referencia reales.

-X: distancia entre los objetos de referencia representados (en pixeles).

-h_1: distancia focal de la cámara indicada por el fabricante, 3.85mm.

Y mediante las siguientes ecuaciones se puede hallar el valor del pixel:

$$\frac{h1}{h2} = \frac{n1}{n2}$$

$$n_1 \text{ x valorPixel} = X$$

Para obtener valores, se ha tomado como referencia para cada cámara tres posiciones distintas de los objetos reales, con lo que se ha conseguido un valor medio para ambas cámaras de $13.78 \text{x} 10^{-6}$m/pixel.

Parámetros extrínsecos

Los parámetros extrínsecos relacionan los sistemas de referencia del mundo real y la cámara describiendo la posición y orientación de la cámara en el sistema de coordenadas del mundo real.

Inclinación

Se considera inclinación de una cámara al ángulo que ésta forma con el eje Z del mundo real.

A pesar de tener las cámaras distintos ejes locales, la inclinación con respecto a las coordenadas locales son las mismas, por lo que para explicar la calibración de la inclinación de las cámaras se utilizarán gráficos distintos, pero las mismas ecuaciones.

Para el cálculo de la inclinación se plantea la siguiente situación:

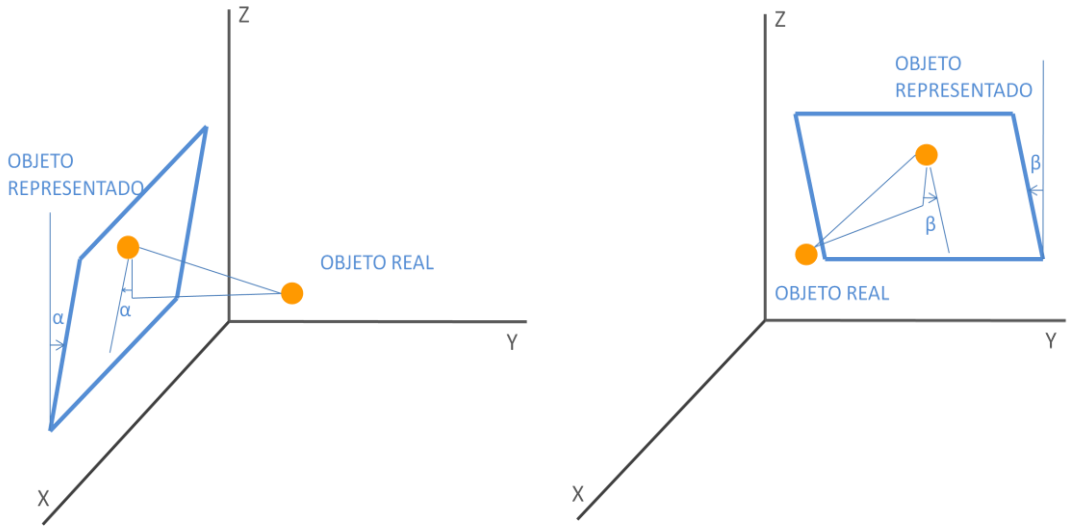


Figura 14: Inclinación en ambas cámaras

Que, de perfil, queda así:

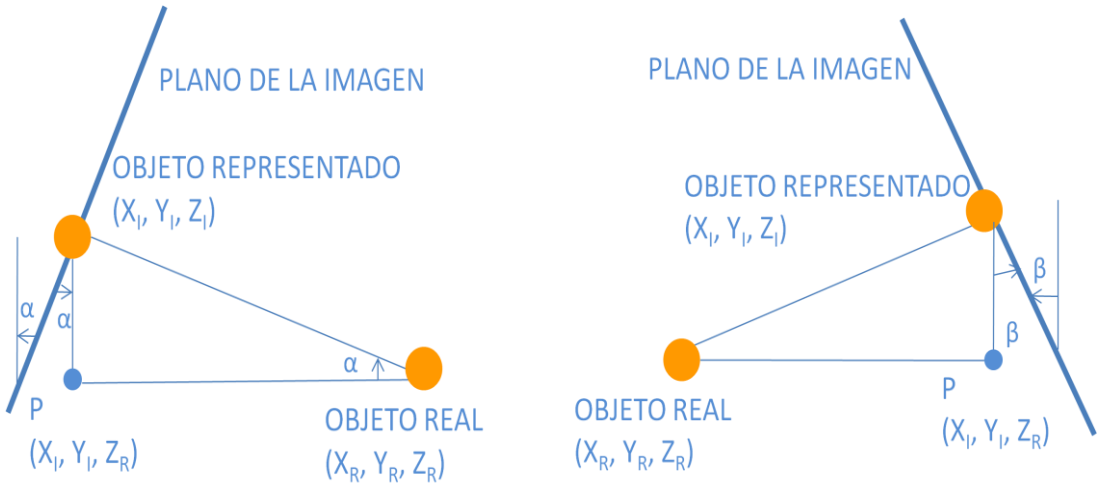


Figura 15: Esquema para el cálculo de la inclinación en ambas cámaras

Donde Objeto real se conoce y Objeto representado se obtiene a partir de la imagen. El punto P está en la recta paralela al plano XZ en el caso de la cámaraXZ y al plano YZ en caso de que sea la cámaraYZ, que pasa por el Objeto representado.

De los esquemas anteriores podemos obtener, en consecuencia:

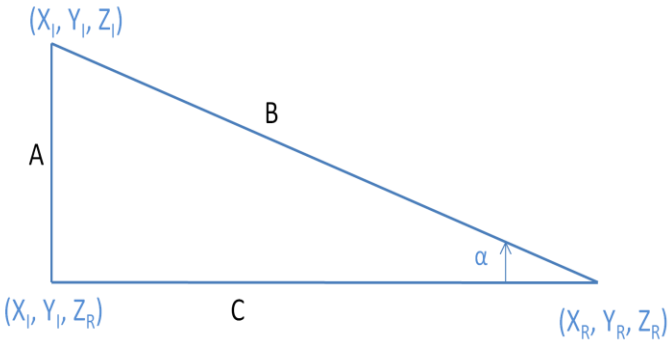


Figura 16: Esquema para el cálculo de la inclinación

Con lo que se puede calcular el ángulo de inclinación de cualquiera de las cámaras, α, así:

$$sen \propto = \frac{A}{B}$$

con

A = distancia $((X_I, Y_I, Z_I), (X_I, Y_I, Z_R))$

B = distancia $((X_I, Y_I, Z_I), (X_R, Y_R, Z_R))$

Distancias de los ejes

Las cámaras pueden estar desplazadas en su plano con respecto a los ejes y por tanto, no estar situadas en el origen de coordenadas del mundo real, por ello el usuario debe conocer las distancias de las cámaras a los ejes del mundo real.

La cámaraXZ está situada en el plano XZ, pudiendo estar posicionada en cualquier punto de ese plano desplazada a través de los ejes X y Z del origen de coordenadas del mundo real una distancia siempre positiva.

La cámaraYZ está situada en el plano YZ. Ésta puede estar desplazada del origen de coordenadas a través de los ejes Y y Z una distancia positiva.

Matriz de transformación

Debido a la inclinación y a las distancias de ambas cámaras con respecto a los ejes del mundo real, es necesario hallar sus matrices de transformación para pasar de coordenadas locales de una cámara a coordenadas del mundo real.

Estas matrices de transformación tienen como parámetros el ángulo de inclinación de las cámaras con respecto al eje Z real (α para la cámaraXZ y β para la cámaraYZ) y la distancia de las cámaras a los ejes del mundo real (en el caso de la cámaraXZ, d1 es la distancia en el eje X y d2, en el eje Z; en el caso de la cámaraYZ, d1 es la distancia en el eje Y y d2, en el eje Z)

Matriz de transformación de la cámaraXZ $\begin{pmatrix} 1 & 0 & 0 & d1 \\ 0 & sen \propto & -cos \propto & 0 \\ 0 & cos \propto & sen \propto & d2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Los ejes de la cámaraXZ queda, tras la transformación de ejes así:

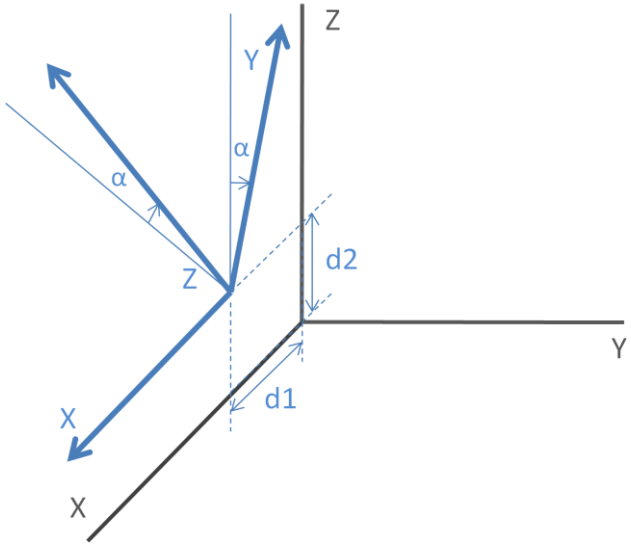


Figura 17: Parámetros extrínsecos de la cámaraXZ

Matriz de transformación de la cámaraYZ $\begin{pmatrix} 0 & sen\beta & cos\beta & 0 \\ 1 & 0 & 0 & d1 \\ 0 & cos\beta & -sen\beta & d2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Los ejes de la cámaraYZ quedan así tras la transformación:

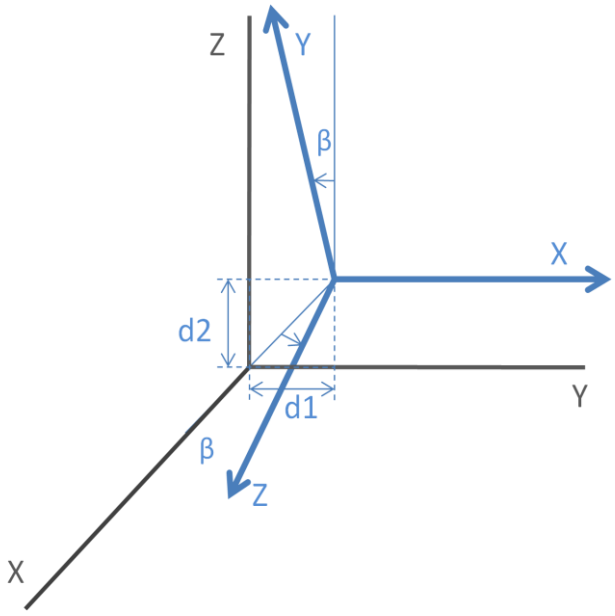


Figura 18: Parámetros extrínsecos de la cámaraYZ

2.3. Entornos de medida

A lo largo del desarrollo del proyecto se han ido utilizando distintos entornos de medida de laboratorio según el nivel de complejidad de las pruebas que se necesitaban realizar para las distintas funcionalidades de la aplicación.

Al comienzo del desarrollo no se requirió la utilización de ningún entorno concreto, ya que las únicas pruebas a realizar eran la comprobación de la captura de las imágenes por medio de las webcams y el filtrado de éstas para, posteriormente, realizar la localización de los objetos en la propia captura.

Una vez comenzado el desarrollo de la funcionalidad de localización del objeto en el mundo real por medio de los datos obtenidos de las dos cámaras, se necesitó el uso de entornos de medida controlados para la realización de las pruebas para así poder verificar la precisión y la exactitud de los resultados. Estos entornos se describen más detalladamente a continuación.

2.3.1. Micromundo

Para realizar las pruebas iniciales del sistema se necesitaba un entorno reducido, que se pudiera montar fácil y rápidamente para realizar las pruebas perdiendo el mínimo tiempo posible en su montaje, pero que sirviera para probar la mayor parte de las funcionalidades del sistema.

Para este propósito utilizamos un entorno con una base de 35x55 cm de color negro y con divisiones cada centímetro, con lo que resultaba sencillo colocar las cámaras y los objetos de referencia en posiciones concretas, pudiendo comprobar los resultados obtenidos fácilmente.

Con este entorno se realizaron las pruebas para la funcionalidad básica del sistema, es decir, el cálculo de la posición y de la orientación, además de las funcionalidades de calibración de la inclinación y de la distancia focal, ya que, aunque se trate de un entorno de dimensiones reducidas, la validez de los resultados para estas funcionalidades es la misma, puesto que tanto este entorno como el de las pruebas reales en un laboratorio son entornos controlados.

De este entorno se obtuvo también la información necesaria para realizar los cálculos para hallar el valor en milímetros de un píxel en las cámaras. Este proceso se ha explicado detalladamente en el apartado *Parámetros de las cámaras* de este capítulo.

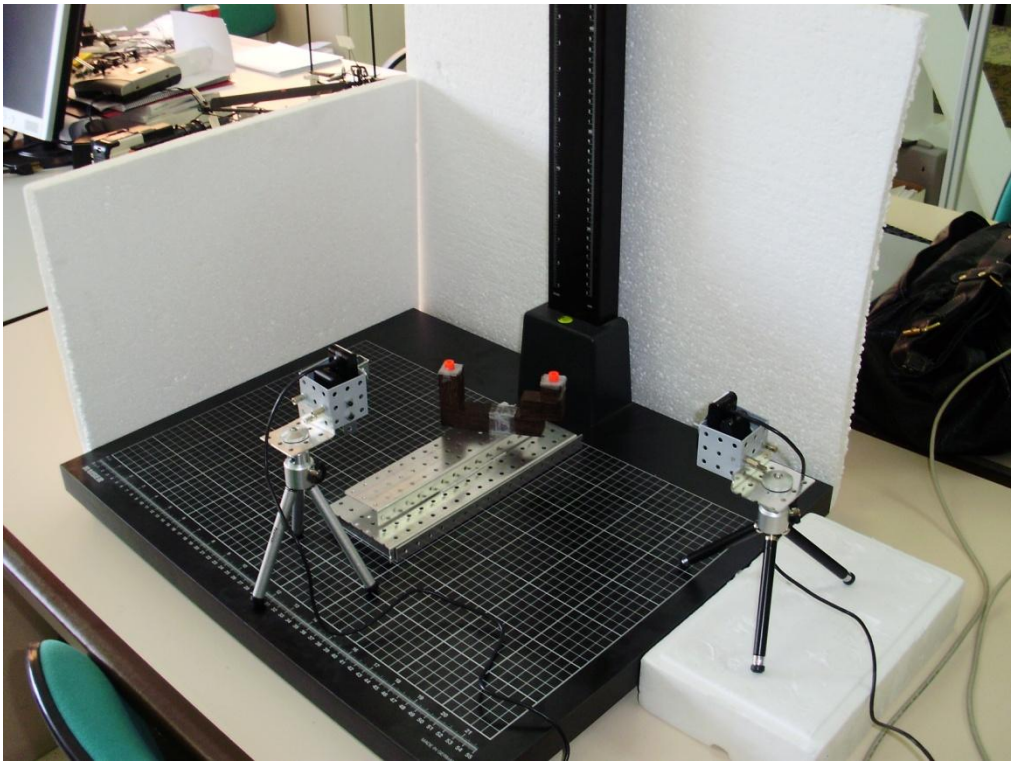


Figura 19: Micromundo

2.3.2. Entorno de laboratorio

Una vez determinado que los resultados obtenidos por el sistema eran fiables se procedió a realizar pruebas sobre un entorno de un tamaño mayor, tratándose de

un entorno controlado de unas dimensiones de aproximadamente 2x2m, usando como objetos de referencia dos bolas de color rojo.

Sobre este entorno se realizaron pruebas de seguimiento de los objetos de referencia realizando su movimiento manualmente y, finalmente, colocando los objetos sobre un helicóptero y un cuatrimotor para simular pruebas reales.

Capítulo 3. Localización

3.1. Introducción

En este apartado se explicará la metodología llevada a cabo para el cálculo de la posición del objeto de referencia y de su orientación en el espacio gracias a las imágenes de dos cámaras.

3.2. Posición

El sistema de localización desarrollado es un sistema de medida en 3D que emplea visión estereoscópica por medio de las imágenes obtenidas de dos cámaras colocadas en planos perpendiculares entre sí.

Con una imagen se obtienen coordenadas 2D de un objeto y por tanto no se puede calcular la profundidad de un objeto en la escena, por lo que con las imágenes procedentes de dos cámaras se obtienen dos pares de coordenadas 2D que permiten calcular las coordenadas 3D del objeto de referencia con respecto al mundo real.

Para la localización del objeto de referencia, por tanto, es necesario tener una captura de cada cámara sobre la que trabajar.

Cada una de las cámaras tiene en su interior una lente convergente que hace que, cuando la distancia del objeto es mayor que la distancia focal de la cámara, la imagen que se crea es real e invertida, debido a que los rayos de luz convergen en el foco, como se puede ver en la siguiente figura.

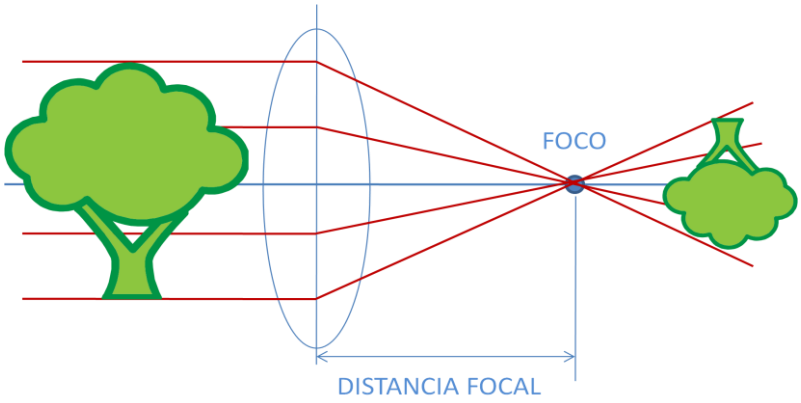


Figura 20: Efecto de la lente convergente de la cámara

Puesto que para las cámaras utilizadas la distancia focal indicada por el fabricante es de 3.85mm, es de suponer que, en ninguno de los casos de uso del localizador, el objeto se encontrará a menor distancia de la dada. Así que, con este supuesto, se

invierte la imagen para que sea tal y como ve el usuario la realidad, es decir, se rota la imagen 180°.

Una vez obtenidas las dos imágenes, se filtra cada una de ellas para aislar el objeto de referencia del resto. Este procesamiento, detallado en el Capítulo 4, consiste en un filtro gaussiano que quita el ruido de la imagen, un filtro de colores que binariza la imagen según la combinación del espacio de color elegida por el usuario y un cierre. Tras el filtrado, se lleva a cabo la localización de los objetos de interés obteniendo de cada imagen un par de coordenadas 2D expresadas en píxeles.

Antes de comenzar la localización, es necesario calibrar las cámaras, y como se ha visto en el capítulo anterior, el tamaño del pixel en milímetros calculado es 13.78×10^{-6}m/pixel. Por ello cada vez que se obtenga, en cualquier paso de la localización, coordenadas 2D expresadas en píxeles, será necesario transformarlas a milímetros para utilizar medidas del mundo real así:

Coordenadas en píxeles x valorPixel = Coordenadas en milímetros

Una vez conocidas las coordenadas en milímetros, se pueden calcular las coordenadas de los puntos en el mundo real.

Para conocer las coordenadas de un punto real con respecto a unos ejes trasladados y rotados mediante la matriz de transformación, es necesario hacer lo siguiente:

Matriz de transformación x Punto real = Punto imagen

Por lo que, para hacer el cálculo inverso, es decir, conociendo las coordenadas en la imagen, hallar las coordenadas con respecto a los ejes reales, es necesario hacer:

Punto real = $Matriz\ transformación^{-1}$ x Punto imagen

Para calcular la posición de un objeto en una imagen, hay que trazar una recta desde el objeto que pase por el foco de la cámara y llegue al plano de la imagen, y en el punto donde la recta y el plano intersecan, se encuentra la representación del objeto en la imagen, como se puede ver en la Figura 21.

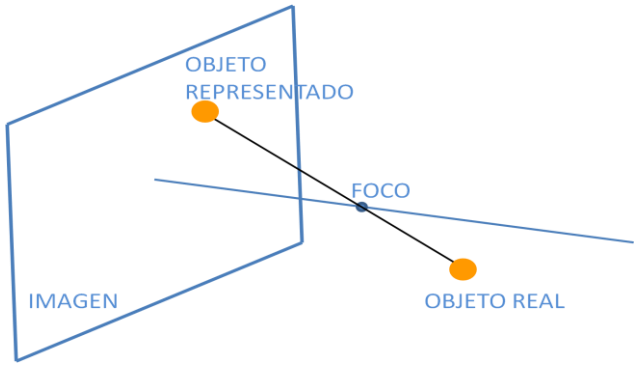


Figura 21: Esquema del foco

Por lo tanto, el localizador se encuentra en la situación opuesta ya que se cuenta con la representación en dos cámaras y los focos de ambas para calcular la posición del objeto real. Es por ello por lo que se lleva a cabo el cálculo inverso. Es decir, una vez conocidos los focos de las cámaras, se crean las rectas desde cada uno de ellos al punto de la cámara correspondiente, puesto que será en estas rectas donde se encontrará el punto real buscado.

Como el objeto real puede estar en cualquiera de los puntos de las rectas creadas y ya que para la localización se emplean dos cámaras, será en la intersección de las rectas de una cámara con las rectas de la otra donde se obtendrá la solución, como se observa en la siguiente figura:

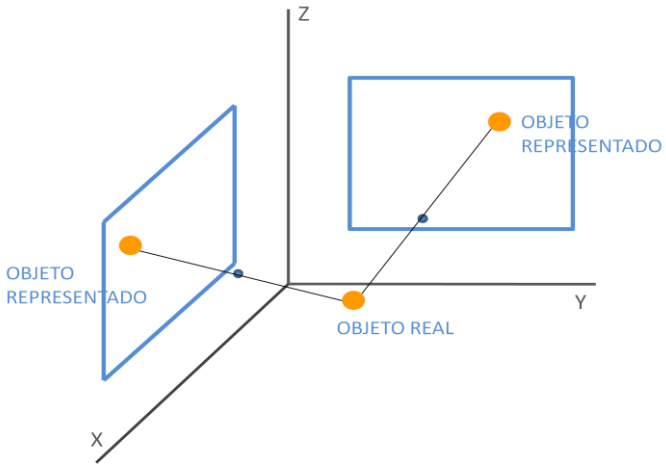


Figura 22: Localización del objeto real mediante las imágenes de las dos cámaras

Cabe destacar que la opción de que las rectas sean paralelas no es posible y además no aportaría información ninguna al cálculo de la posición del objeto puesto que la distancia entre las dos rectas es siempre la misma y cualquier pareja de puntos podría ser solución.

Además, debido a pequeños errores producidos por imprecisiones en la estima del punto en la imagen o de los parámetros de la cámara, no siempre se producirá una intersección exacta entre las rectas, es decir puede ocurrir que las rectas se corten o se crucen.

En caso de que las rectas se corten, caso poco probable por los errores comentados anteriormente, la solución sería inmediata puesto que consistiría en la intersección de las dos rectas dando lugar a una solución exacta.

En el caso de que las rectas se crucen, se buscará la pareja de puntos (uno de cada recta) entre la que haya una distancia menor, considerando que esta distancia es fruto de pequeñas desviaciones en las rectas producidas por los errores comentados. Así, se conseguirá un punto solución muy aproximado a la realidad.

Puesto que el caso más frecuente que se va a dar en el localizador es que dos rectas se crucen, expuesto en la Figura 23, se calcula la distancia mínima entre las dos rectas de la siguiente manera.

Las rectas R1 y R2 son las rectas calculadas que pasan por la representación del objeto y por el foco en cada una de las cámaras. Estas rectas están representadas con las ecuaciones paramétricas de la recta, mediante un punto de la recta y un vector director.

R1: $P(p_x, p_y, p_z)$ $\vec{v}(v_x, v_y, v_z)$

R2: $Q(q_x, q_y, q_z)$ $\vec{w}(w_x, w_y, w_z)$

Distancia mínima (r1, r2) = $\frac{|\overrightarrow{PQ}\,\vec{v}x\vec{w}|}{||\vec{v}x\vec{w}||} = \frac{\begin{vmatrix} vx & wx & qx-px \\ vy & wy & qy-py \\ vz & wz & qz-pz \end{vmatrix}}{||\vec{v}x\vec{w}||}$

Debido a que la solución buscada está a la menor distancia posible de las rectas, se encontrará en la recta perpendicular común a ambas que se corte con las dos.

$\vec{u} = \vec{v}$ x $\vec{w}$ es un vector perpendicular a los vectores directores de las rectas R1 y R2: $\vec{v}$ y $\vec{w}$, y paralelo a la perpendicular común, que servirá para representar la recta perpendicular común a R1 y R2 buscada.

Por otra parte la perpendicular común quedará definida también por los puntos r (r_x, r_y, r_z), perteneciente a R1 y s (s_x, s_y, s_z), perteneciente a R2. Los puntos r y s quedan definidos en función de la recta a la que pertenece cada uno de ellos:

$(r_x, r_y, r_z) = (p_x, p_y, p_z) + \lambda\,(v_x, v_y, v_z)$

$(s_x, s_y, s_z) = (q_x, q_y, q_z) + \mu\,(w_x, w_y, w_z)$

Y por último, la recta que pasa por r y s y cuyo vector director es $\vec{u}$ queda definida así:

$(s_x, s_y, s_z) = (r_x, r_y, r_z) + t\,(u_x, u_y, u_z)$

De esta manera queda planteado un sistema de ecuaciones así:

$(r_x, r_y, r_z) = (p_x, p_y, p_z) + \lambda\,(v_x, v_y, v_z)$

$(s_x, s_y, s_z) = (q_x, q_y, q_z) + \mu\,(w_x, w_y, w_z)$

$(s_x, s_y, s_z) = (r_x, r_y, r_z) + t\,(u_x, u_y, u_z)$

Planteado el sistema y resuelto, se obtienen los valores de t, μ y λ. μ y λ permiten conocer los puntos r y s de las rectas R1 y R2, respectivamente, que serán la pareja de puntos a menor distancia.

Como se ha explicado anteriormente, la solución al problema de la localización que se puede encontrar más aproximada, estará en la posición más cercana a las dos rectas posible, así que se calcula como la media de r y s.

$$(solucion_1, solucion_2, solucion_3) = ((r_x, r_y, r_z) + (s_x, s_y, s_z))/2$$

En la siguiente figura se puede observar un ejemplo del caso en que las rectas se crucen.

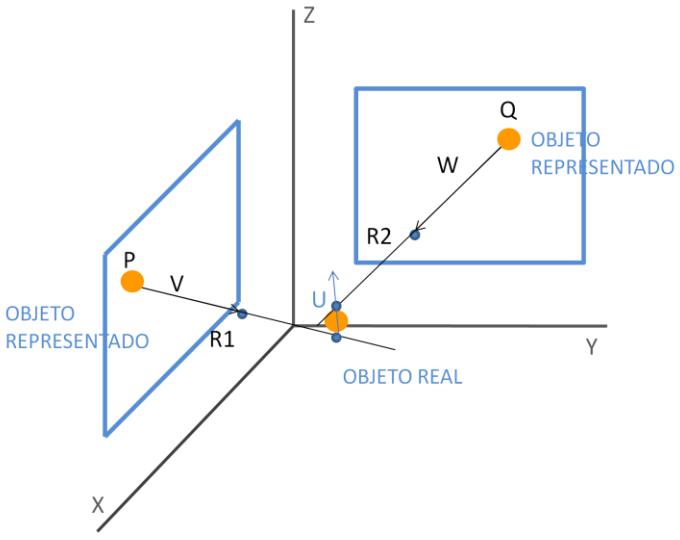


Figura 23: Cálculo del cruce de las dos rectas focales

Como resultado del error anteriormente explicado, se propaga un pequeño error haciendo que la posición obtenida como resultado del proceso de la localización no sea exactamente la posición real del objeto de referencia.

3.3. Orientación

En este apartado se mostrarán los pasos a seguir para el cálculo de la orientación del objeto de referencia.

La orientación permite conocer la dirección del objeto que se toma como referencia para que, una vez calculada la posición, se pueda tener conocimiento de hacia dónde debe dirigirse el objeto. Es decir, en caso de emplear el sistema de localización con un objeto móvil, no es suficiente conocer la posición puesto que, si se necesita que el objeto se mueva en una dirección determinada distinta a la que está colocada, la información es insuficiente para intentar controlar el objeto.

Pero, debido al uso de un solo color para poder determinar el objeto de referencia se conocerá la posición y la orientación del objeto (entre 0° y 180°), es decir, su dirección, pero no el sentido hacia el que está dirigido.

Esto se ha diseñado así debido a que el sistema está pensado para realizar un control de la posición, por lo que una vez obtenido el valor inicial del vector dirección los cambios que se van a producir en el ángulo van a ser progresivos, desechándose los valores que impliquen un cambio brusco en la dirección.

Para el cálculo de la orientación, en primer lugar, se obtiene el vector formado por los dos puntos de referencia del objeto (p y q) así:

$$(v_x, v_y, v_z) = (p_x, p_y, p_z) - (q_x, q_y, q_z)$$

Se calcula siempre el ángulo relativo al plano XY, es decir, se tendrá en cuenta el ángulo formado por la proyección de este vector $\vec{v}$ sobre el plano XY, por ello sólo se empleará en el cálculo v_x y v_y:

$$\text{Ángulo} = \cos^{-1} \frac{|vx|}{\sqrt{vx^2 + vy^2}} * 180.0/\pi$$

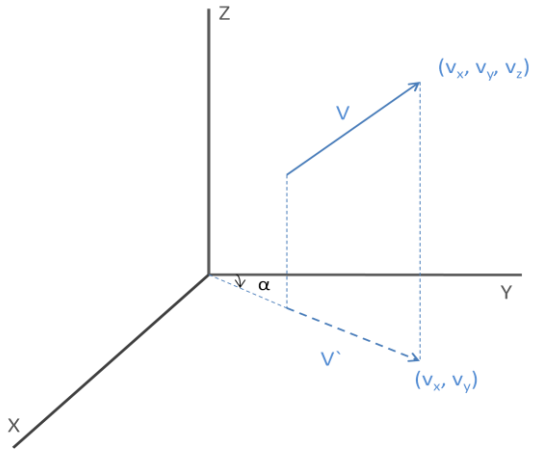


Figura 24: Cálculo de la orientación

Se toma como referencia el eje X, de forma que en el plano XY cuando el vector obtenido está en el primer y tercer cuadrante, obtenemos el ángulo buscado, sin embargo, en el caso opuesto, obtenemos el ángulo suplementario del buscado. Por ello, para obtener un ángulo en el rango (0°, 180°), cuando el vector está en el segundo o cuarto cuadrante, se hace:

ángulo = 180 – ángulo

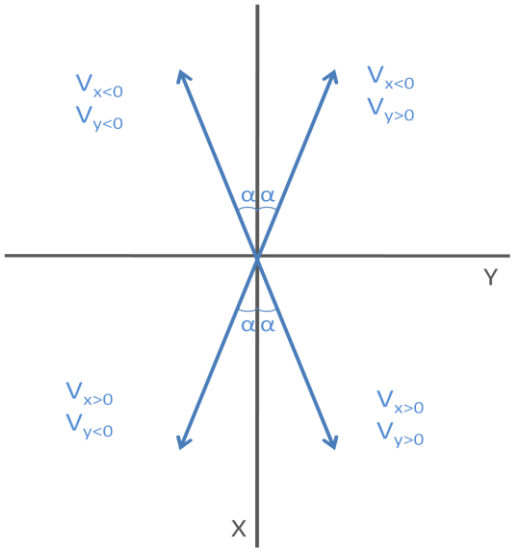


Figura 25: Ángulo deseado para el cálculo de la orientación

Capítulo 4. Diseño Software

4.1. Introducción

Para el desarrollo de la aplicación se han utilizado librerías y software de código abierto.

El sistema de localización se ha desarrollado en lenguaje C++, utilizando para ello como compilador la suite MinGW, y como entorno de desarrollo eclipse junto con la herramienta CDT que permite utilizar este entorno, por defecto para Java, para programar en C/C++.

Para la captura y el tratamiento de imágenes se ha utilizado la librería de código abierto OpenCV. Esta librería proporciona un marco de trabajo de alto nivel para el desarrollo de aplicaciones de visión por computador en tiempo real: estructuras de datos, procesamiento y análisis de imágenes, análisis estructural, etc. Este marco de trabajo facilita en gran manera la implementación de distintas técnicas de visión por computador, aislando al desarrollador de las peculiaridades de los distintos sistemas de visión.

Todo este tratamiento está incluido en una DLL que se ha diseñado para un uso más variado de la parte de visión. La librería incluye todo lo necesario tanto para la localización de los objetos como para la calibración de ciertos aspectos de las cámaras.

Una vez conseguida una versión funcional del sistema, se ha utilizado el IDE Visual Studio para desarrollar una interfaz más amigable para la aplicación.

También se ha desarrollado una aplicación Java, utilizando el IDE Netbeans, que se encarga de recibir los datos a través de UDP y dibuja la posición en la que se encontraría el objeto en cada instante de la ejecución.

A continuación se describe el funcionamiento del sistema de localización.

4.2. Descripción del sistema

El sistema de medida tiene cuatro fases bien diferenciadas:

- Captura de las imágenes.
- Procesamiento de las imágenes.
- Localización de los objetos en las imágenes.
- Cálculo de la posición y la orientación.

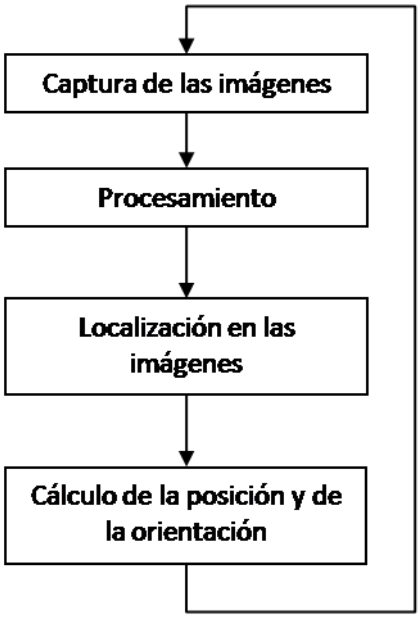


Figura 26: Diagrama de la librería

Para una mayor versatilidad estas fases se han desacoplado del interfaz, consiguiendo así que toda la funcionalidad del programa esté encapsulada en una DLL. Esta librería puede ser introducida en cualquier programa, dotando a este de un sistema de posicionamiento integrado. Otra posibilidad es que el usuario pueda diseñar su propio interfaz gráfico si el nuestro no se adapta a sus necesidades. Además del motor se ha diseñado un par de interfaces: una versión de ventanas más completa y con un aspecto más agradable pero con "mayor peso" y una versión DOS (en modo consola) más "ligera".

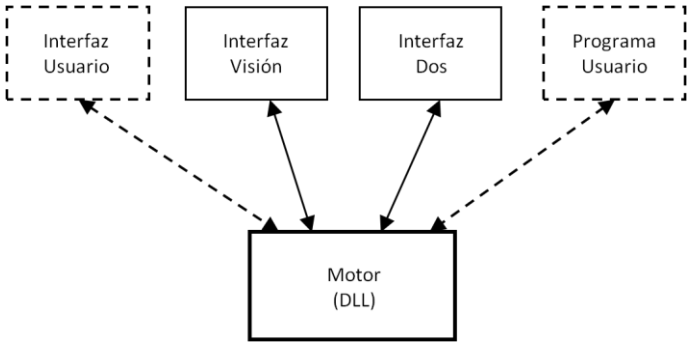


Figura 27: Diagrama del sistema

Las funciones disponibles en la DLL se encuentran detalladas en el Apéndice A *Documentación de la DLL.*

4.2.1. Captura

Esta fase consiste en la captura de las imágenes a través de las webcams y su almacenamiento en memoria para posteriormente realizar su tratamiento. Además, este proceso debe realizarse lo más rápidamente posible, ya que el sistema necesita enviar la mayor cantidad de información por unidad de tiempo.

4.2.1.1. Prototipo

Para el proceso de captura de este sistema utilizamos las funciones que proporciona OpenCV. Existen dos maneras de realizar la captura de imágenes mediante esta librería, utilizando las funciones del módulo *cvcam* o las del módulo *highgui*. La principal diferencia y por la cual nos decantamos por las funciones proporcionadas por el módulo *cvcam* es que este permite la ejecución de visión estéreo con dos cámaras, mientras que con el módulo *highgui*, pese ser más sencilla su configuración, la captura con las dos cámaras debe realizarse de manera iterativa, es decir, una a continuación de la otra, por lo que se pueden dar problemas de correspondencia entre las imágenes, mientras que con *cvcam*, al capturarse las imágenes a la vez este problema no se da.

Los pasos necesarios para realizar la captura por medio de las funciones del módulo *cvcam* son los siguientes:

- Se crean las 4 ventanas para mostrar las dos imágenes originales de ambas cámaras y sus correspondientes imágenes después del procesado. Las ventanas se crean con la función *cvNamedWindow*.
- Se crean las imágenes que se van a utilizar para el procesamiento de las capturas con diversas llamadas a la función *cvCreateImage*.
- Configurar la captura de cada una de las cámaras mediante llamadas a la función *cvcamSetProperty* para asociar a cada cámara una ventana en la que mostrar su captura, el tamaño de la imagen obtenida y la llamada a la función que se encargará del procesamiento de las imágenes.
- Se inicializan las capturas mediante la llamada a *cvcamInit* y comienza la ejecución con la llamada a *cvcamStart*.
- Esperar la pulsación de la tecla para detener la captura.
- Se detiene la ejecución mediante la llamada a *cvcamStop* y liberamos todos los recursos con *cvcamExit*.
- Se libera la memoria eliminando las imágenes con la función *cvReleaseImage* y las ventanas con *cvDestroyWindow*.

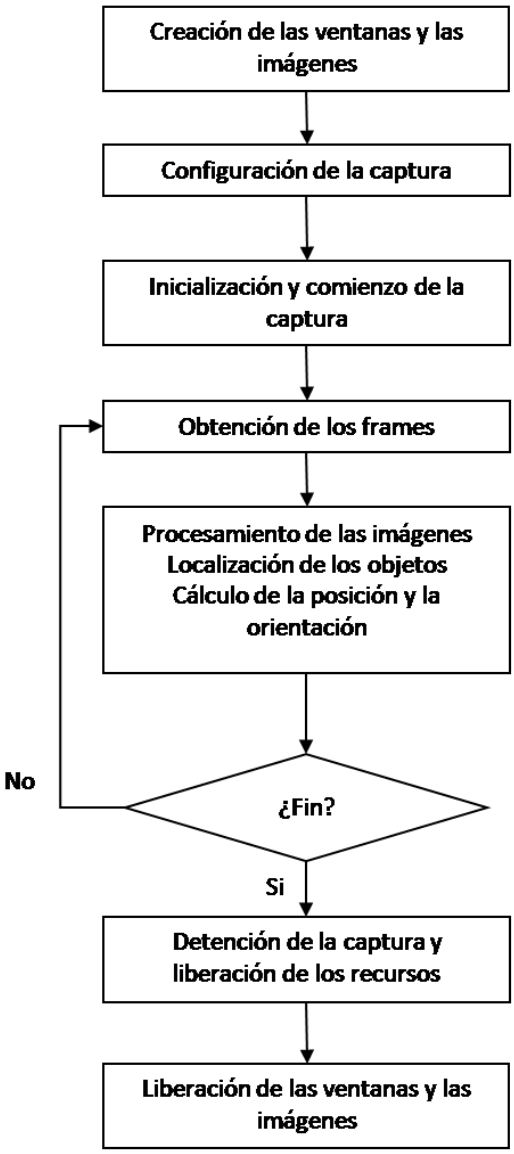


Figura 28: Diagrama de la captura

Este método de captura funciona correctamente con la interfaz desarrollada con los módulos que proporciona OpenCV, pero al intentar integrarla con otra interfaz más compleja utilizando la librería Qt tuvimos muchos problemas, ya que este método de captura unido a la interfaz de Qt consumía muchos recursos, llegando a bloquear la ejecución de la aplicación. Por esta razón se tuvo que detener esa línea de trabajo y se buscaron otras alternativas, descubriendo un método de captura realizado para la herramienta Visual Studio, con el que se realizaron diversas pruebas satisfactorias. El método de captura que finalmente se desarrolló se describe a continuación.

4.2.1.2. Versión final del sistema

Para el proceso de captura de este sistema se ha utilizado la librería videoInput, la cual permite una captura de imágenes rápidas. Esta librería funciona mejor que las de OpenCV, que daban problemas de sincronización y era prácticamente imposible incrustarlas en cualquier GUI que no fuera la propia de OpenCV.

La captura con este método es asíncrona. Se tiene un hilo dedicado a la captura de las imágenes cada vez que las cámaras dan un frame nuevo y su posterior procesamiento.

El método de captura se ha dividido en tres fases:

- Proceso de inicialización: en el cual se inicializan las cámaras y el hilo de captura.
- Proceso de captura: durante este proceso de capturan las imágenes, se procesan, se envían al GUI y se obtienen las coordenadas procedentes del modulo localización.
- Proceso de terminación: se para el hilo, se desactivan las cámaras y se liberan recursos.

A continuación se pasa a detallar los pasos para cada una de estas fases.

Proceso de inicialización

Lo primero que se hace en el proceso de inicialización es instanciar el objeto de captura. Para ello se deberá haber definido previamente una variable del tipo *videoInput* e instanciarlo creando un objeto de esta clase, con el comando *new videoInput ()*.

El siguiente paso es obtener el número de cámaras con la orden *listDevices()* sobre el objeto de la clase *videoInput*.

Después hay que activar cada una de las cámaras sobre el objeto antes mencionado con la orden *setupDevices(i)*, donde i se refiere al número de cámara. Primero se instancia la primera cámara (i=0) y después la segunda (i=1). A Continuación se manda iniciar el módulo correspondiente.

Lo último que se hace es lanzar el hilo de captura con la función propia del API de Windows *createTread*.

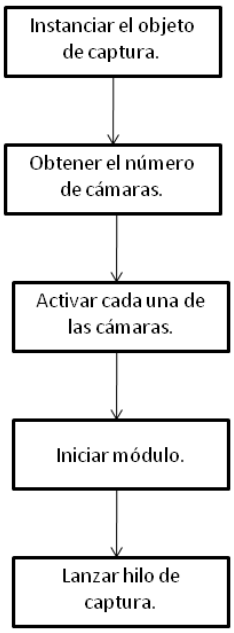


Figura 29: Diagrama del Proceso de Inicialización de la Captura

Proceso de captura

Este proceso se ejecuta mientras no se decida terminar con el hilo (poniendo a *true* la variable de terminación, con lo cual se sale del bucle, véase proceso de terminación). Se va capturando imágenes y procesándolas.

Se pregunta a la primera cámara si tiene una nueva imagen con el comando *isFrameNew(0)* sobre el objeto de la clase videoInput. Si se tiene una nueva captura se continúa preguntando a la cámara 2. Si no hay imagen nueva se vuelve a la comprobación de terminado.

Se pregunta a la segunda cámara si tiene una nueva imagen con el comando *isFrameNew(1)* sobre el objeto de la clase videoInput. Si se tiene una nueva captura se capturan las imágenes de ambas cámaras y se continúa con el procesamiento. Si no hay imagen se vuelve a la comprobación de terminado.

Se procesan las imágenes como se explicara a continuación.

Se pasan las imágenes al GUI (o programa que esté recibiendo los datos) mediante la orden *procesador(imagen, identificador)* que es la única que debe implementarse de manera obligatoria en la aplicación que use la DLL (Apéndice A).

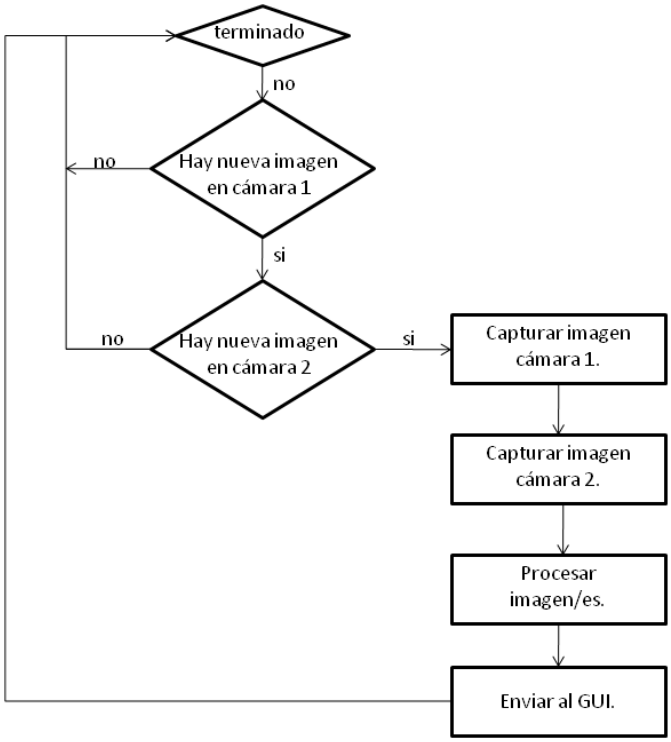


Figura 30: Diagrama del Proceso de Captura

Proceso de terminación

Para terminar con el proceso de captura anteriormente dicho lo primero que hay que hacer es terminar el hilo de captura poniendo la variable *terminación* activa. Hay que esperar a que el hilo termine con la orden del API de Windows *waitForSingleObject*.

Posteriormente hay que desactivar cada una de las cámaras con la función *stopDevices(i)* que se usa como la función de activar las cámaras.

El siguiente paso es liberar recursos que hayamos estado utilizando.

Antes de terminar se guarda el XML con las opciones que hayamos cambiado si el usuario no ha dicho lo contrario.

Para finalizar se termina el módulo que se esté ejecutando: modulo localización, calibración de la distancia focal o calibración de la inclinación.

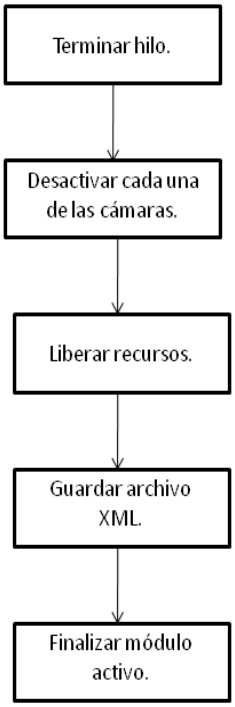


Figura 31: Diagrama del Proceso de Terminación de la Captura

4.2.2. Procesamiento

Las imágenes obtenidas mediante el proceso de captura deben ser tratadas con el objetivo de conseguir una imagen binarizada para que, durante la fase de localización, sea posible encontrar los objetos de referencia.

Esta fase ha sufrido diversos cambios a lo largo del desarrollo. Al principio se implementó de manera que tras binarizar la imagen se calculaban los bordes de los objetos, y al tener forma circular al tratarse de bolas, se intentó aplicar una función que buscaba circunferencias y obtenía sus coordenadas en la imagen. Este método se descartó ya que se vio que era muy complicado obtener circunferencias perfectas de los objetos filtrados, ya que las sombras y el propio aparato en el que se encontraban colocadas las bolas cortaban la forma. Tras observar esto se decidió descartar la obtención de los bordes del objeto tras la binarización y se siguen los siguientes pasos:

- El primer paso que se realiza es voltear las imágenes para que al mostrarse por pantalla sea más fácil su seguimiento. Este volteado se consigue utilizando la función *cvConvertImage* que proporciona el módulo *highgui* de la librería de OpenCV.

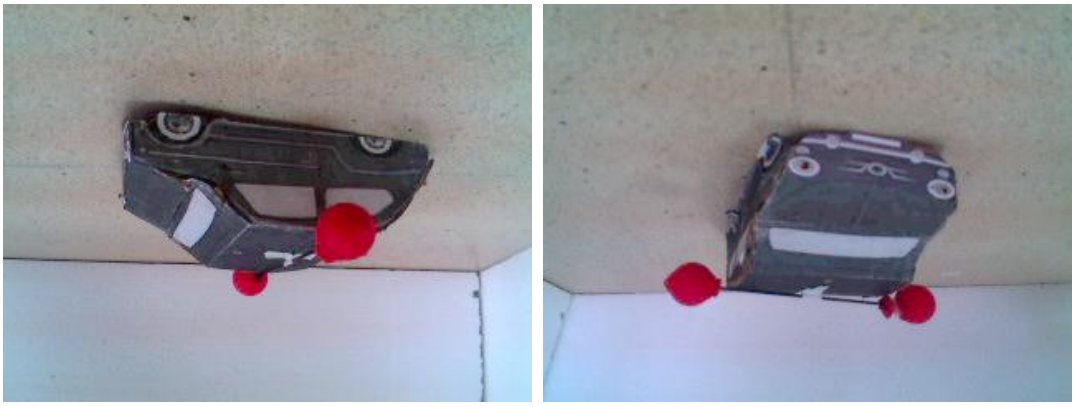


Figura 32: Capturas obtenidas por las cámaras

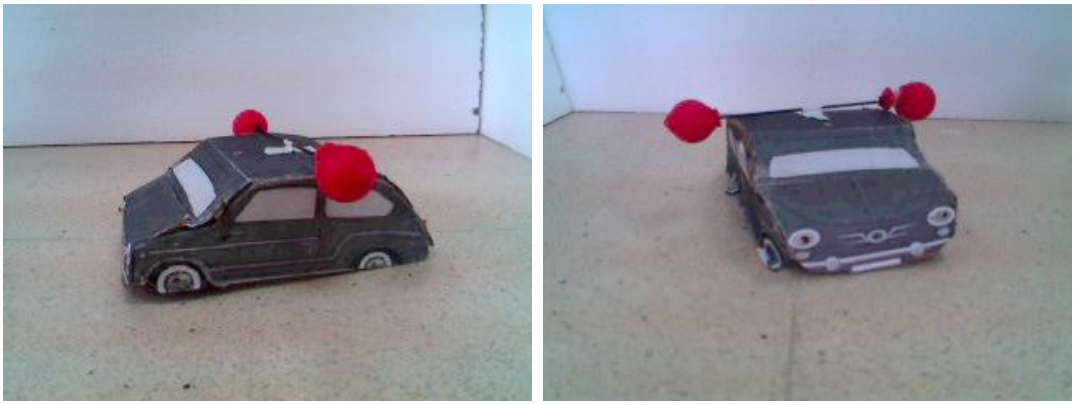


Figura 33: Capturas después del volteado

- Si se está trabajando con los espacios de color HSV o HLS es necesario transformar las imágenes del espacio de color RGB mediante la función *cvCvtColor* que proporciona el módulo *cv*.
- Después de modificar el espacio de color se realiza un filtrado Gaussiano de radio 3 para eliminar el posible ruido obtenido en la digitalización. Para realizar este filtrado se llama a la función *cvSmooth* que proporciona el módulo *cv* de la librería OpenCV.

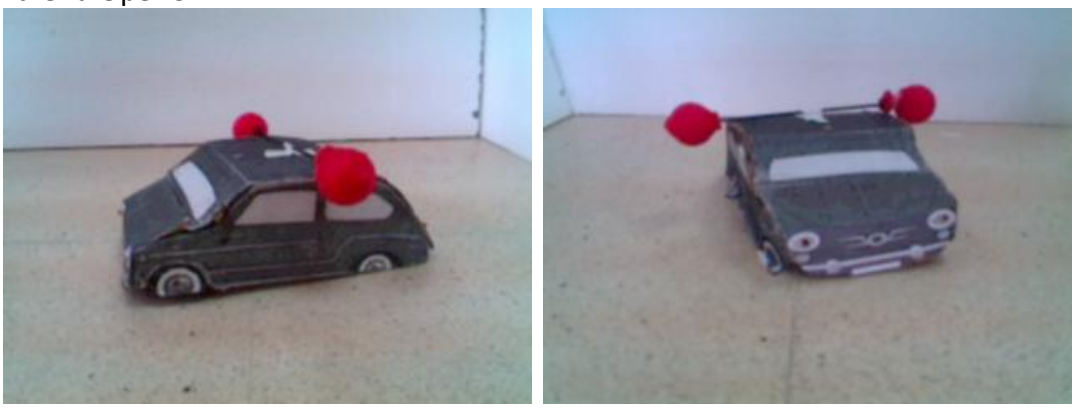


Figura 34: Capturas después del filtrado Gaussiano

- Una vez llegados a este punto binarizamos las imágenes, dejando en blanco la parte de la imagen que se corresponda con el color seleccionado y en negro el resto. Para realizar este binarizado OpenCV no ofrece ninguna función, por lo que ha sido necesario implementar 3 funciones que lo realizan, una por cada espacio de color.

Figura 35: Capturas después del binarizado

- El último paso es realizar un cierre (dilatación + erosión) sobre las imágenes binarizadas utilizando las funciones *cvDilate* y *cvErode* que nos proporciona el módulo *cv* de la librería OpenCV. Para realizar estas operaciones se utiliza la siguiente máscara:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

  Con esto conseguimos rellenar vacíos en el contorno y rellenar agujeros pequeños con el objetivo de que los huecos que aparezcan dentro de los objetos de referencia desaparezcan y sea más sencilla su localización.

Figura 36: Capturas después del cierre

Una vez acabada esta parte, las imágenes están listas para realizar la localización de los objetos de referencia en ellas.

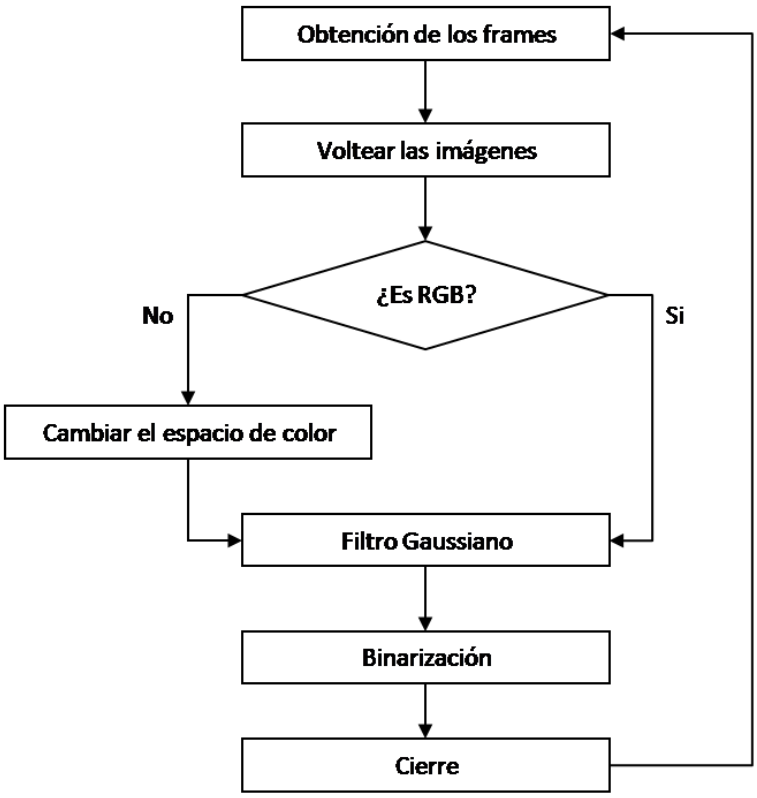


Figura 37: Diagrama del procesamiento de las imágenes

4.2.3. Localización

Después del procesamiento de las imágenes se pasa a la localización de objetos en ellas. Para la localización de objetos en las imágenes se ha usado la búsqueda de clústeres. Un clúster lo tomamos como un conjunto de pixeles del mismo tono que están contiguos.

Recorriendo la imagen de izquierda a derecha y de arriba abajo se busca el primer pixel blanco. Cuando se encuentra uno se introduce en un clúster *c1* y se hace una búsqueda en forma de abanico descendente buscando pixeles contiguos alrededor de este. Cada pixel *pi* que se encuentra se introducen en el clústeres *c1* y se pone en negro en la imagen para que no vuelva a ser procesado. Se vuelve a hacer esta búsqueda para todos los pixeles pi encontrados en el paso anterior, repitiendo el proceso hasta que no se encuentre ningún pixel blanco contiguo a otro en *c1*. Cuando se acaba el proceso se hace la media de las coordenadas de los pixeles para encontrar las coordenadas del clúster *c1.*

Este clúster se mete a una lista de clústeres *c* que posteriormente será utilizada para sacar la posición y orientación real de los objetos.

Posteriormente se han añadido mejoras al algoritmo que se detallan a continuación.

La imagen siguiente muestra como son procesados los clústeres y en el orden en que son introducidos en C según su numeración.

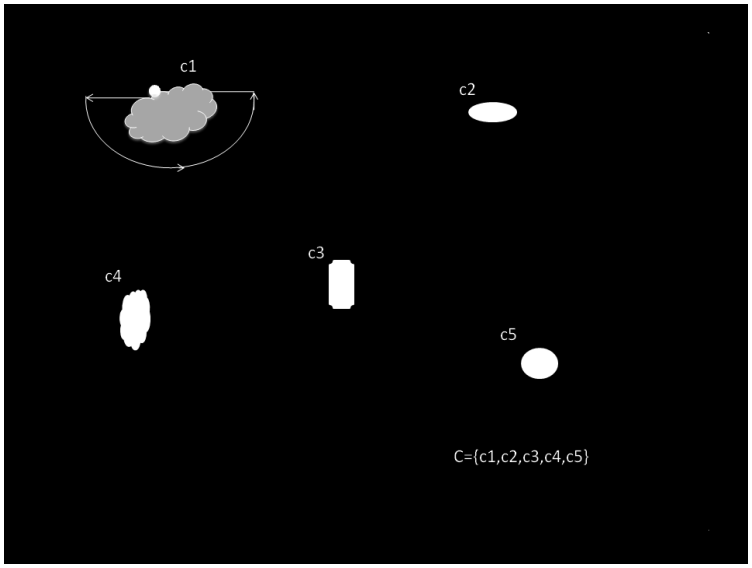


Figura 38: Esquema del proceso de búsqueda de clústeres

```
para todas las filas de la imagen
      para todas las columnas de la imagen
            si el pixel pi[fila, columna] es blanco
            {
                  crear un clúster ci
                  introducir en pixel pi en ci
                  poner el pixel pi en negro
                  mientras haya un pixel contiguo a otro pixel que
                        esté en ci
                  {
                        si el pixel pj es contiguo a alguno en ci
                        {
                              introducir en pixel pj en ci
                              poner el pixel pj en negro
                        }
                  }
                  añadir las coordenadas del clúster ci a la lista
                        de clústeres c
             }
```

En la imagen posterior (Figura 39) se puede observar un conjunto de clústeres de los cuales se ha sacado la media aritmética indicándola con un cuadrado blanco. Esta imagen se refiere a una búsqueda con modo multipunto, el resto de figuras de este apartado se refieren a búsquedas no multipunto.

Hay dos formas de devolver el conjunto de clústeres dependiendo del modo que se esté usando (multipunto o no) y del número de clústeres encontrados:

- El modo no multipunto: en el cual se espera encontrar dos únicos clústeres en cada una de las imágenes. Este módulo pasa las coordenadas de esos dos clústeres al modo de localización. De esta forma se permite obtener el punto donde está el objeto y la orientación de este.
- El modo multipunto: en el cual da igual el numero de clústeres que se vean, se hace la media aritmética de todos los clústeres encontrados y se pasa al modulo de localización. Con este modo solo se conoce la posición del objeto en el mundo real.

Cuando en el modo no multipunto se encuentra un número de clústeres distinto de 2 en ambas imágenes se envía la media aritmética de los clústeres y no se puede conocer la orientación.

Figura 39: Captura búsqueda de clústeres multipunto

4.2.3.1. Mejoras en el algoritmo de búsqueda de clústeres

Búsqueda de regiones

Para la búsqueda no multipunto se ha añadido la posibilidad de búsqueda por regiones. En el caso de qué en una imagen solo se encuentre dos clústeres las siguientes búsquedas se hacen en regiones más pequeñas alrededor de los clústeres que engloban a cada uno de estos (véase Figura 40). El uso de estas regiones hace más efectivas las búsquedas, ya que el tamaño se reduce, y permite que pixeles blancos fuera del clúster se ignoren.

Las regiones de búsqueda vienen delimitadas por un punto (pto1/pto2), la anchura (an1/an2) y la altura (al1/al2) de cada región. El punto se determina a partir del

punto del clúster, la altura(al/2) y la anchura (an/2). Estas pueden ser dinámicas (dependiendo del tamaño del clúster) o las puede definir el usuario.

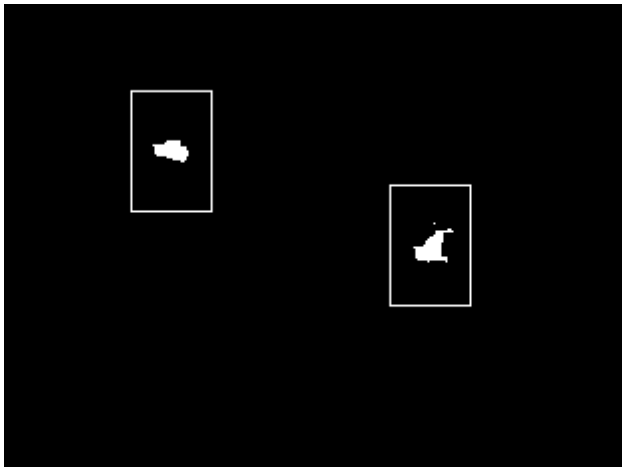


Figura 40: Captura búsqueda de clústeres por regiones

Tamaño mínimo del clúster

Para eliminar pequeños ruidos está la posibilidad de establecer un área mínimo de clúster. Los clúster encontrados de menos tamaño se descartan.

En la Figura 41 se muestran clústeres que no tienen el área mínimo por lo que son descartados.

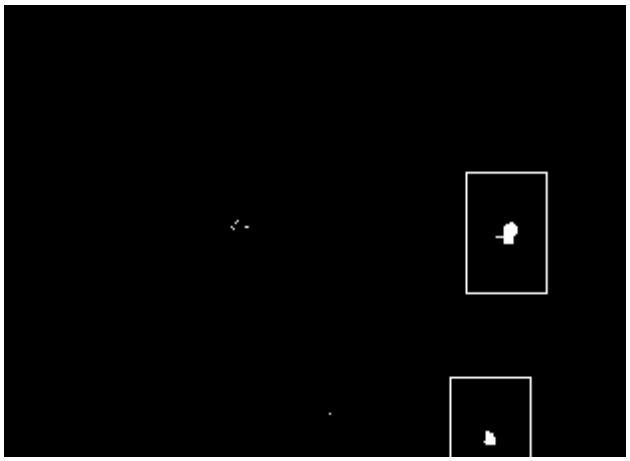


Figura 41: Captura búsqueda de clústeres teniendo en cuenta el área mínimo

Distancia entre clústeres

Como a veces el objeto se ve dividido en varios clústeres por reflejos o sombras se introduce esta opción. Con la cual se permite que, si varios clústeres están a una distancia menor de la indicada, se procesen como un único clúster. Para obtener la distancia entre dos clústeres se usa la distancia euclídea.

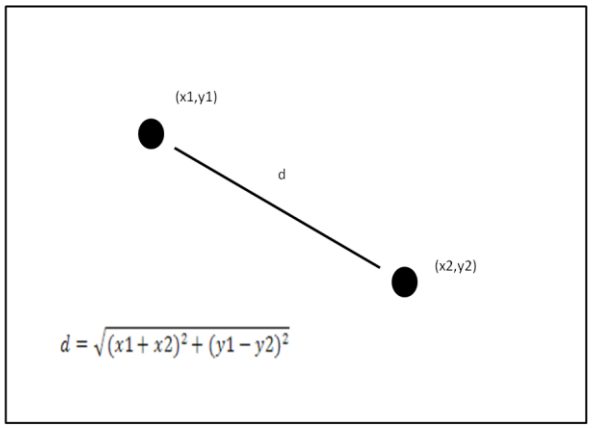


Figura 42: Diagrama de la distancia euclídea

En la figura posterior se muestra dos clústeres que son procesados como un único clúster (región de la derecha).

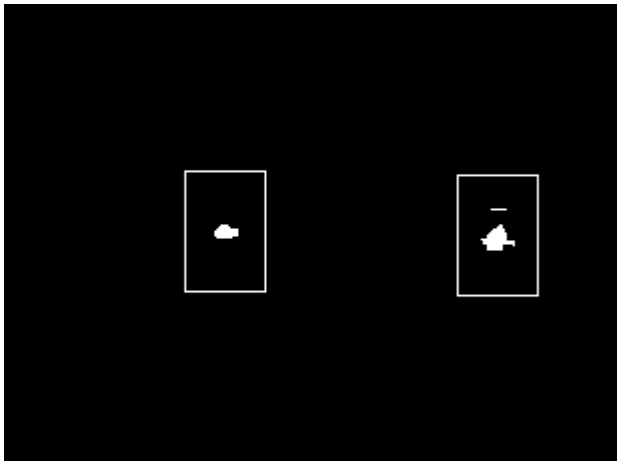


Figura 43: Captura búsqueda de clústeres teniendo en cuenta la distancia entre clústeres

4.2.4. Cálculo de la posición y la orientación

Conocidas las coordenadas de los objetos de referencia en las imágenes, el cálculo de la posición y la orientación permite conocer la posición real del objeto de referencia. Para ello, el desarrollo se basa en la geometría proyectiva de las cámaras pin-hole: la proyección de un punto de la escena se obtiene de la intersección de una recta que pasa por este punto y el foco con el plano imagen.

Por ello, es necesario realizar el proceso inverso, es decir, trazar rectas que pasan por las proyecciones de los puntos en las imágenes y los focos de éstas para conocer posibles posiciones del punto real.

Como en este sistema se emplean dos cámaras, será en la intersección de éstas donde se encuentre el punto real.

La orientación de un objeto permite conocer la dirección que tiene éste en la escena. El procedimiento para el cálculo es sencillo, basta con conocer el ángulo del vector formado por los objetos de referencia con respecto al plano XY.

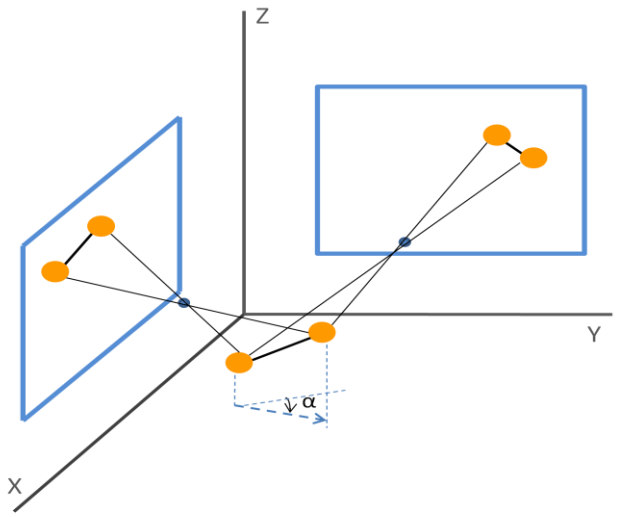


Figura 44: Esquema para el cálculo de la posición y orientación

Los pasos que sigue el sistema son los siguientes:

```
si no se tienen datos suficientes para realizar los cálculo
{
      devolver  punto anterior
      existeOrientacion = false
}
si se obtienen dos parejas de puntos de cada captura
{
      calcular la posición
      existeOrientacion = true
}
si no se obtienen dos parejas de puntos de cada captura
      si está en modo multipunto
      {
            calcular posición
            existeOrientacion = false
      }
      si no está en modo multipunto
            devolver punto anterior
si (existeOrientacion)
      calcular ángulo
si no
      devolver ángulo anterior
```

4.2.5. Configuración mediante XML

Para almacenar los datos de una ejecución a otra de la aplicación se guardan los datos en un archivo xml. En él se guarda toda la información necesaria para que no sea necesario realizar ninguna operación para comenzar a ejecutar la aplicación.

Un ejemplo del xml de configuración sería el siguiente:

```
<?xml version="1.0" ?>
<Configuracion>
  <FicheroConfig Nombre="config.xml" />
  <ModoEjecucion MostrarVentanas="1" Multipunto="0" />
  <Color>
    <EspacioColor ColorSpace="0" />
    <Camara1>
      <RGB>
        <ValorColorRGB R="100" G="100" B="100" />
        <ValorSensibilidadRGB SensibilidadR="40" SensibilidadG="40" SensibilidadB="40" />
      </RGB>
      <HSV>
        <ValorColorHSVMin H="100" S="40" V="40" />
        <ValorColorHSVMax H="140" S="60" V="60" />
      </HSV>
      <HLS>
        <ValorColorHLSMin H="100" L="40" S="40" />
        <ValorColorHLSMax H="140" L="60" S="60" />
      </HLS>
    </Camara1>
    <Camara2>
      <RGB>
        <ValorColorRGB R="100" G="100" B="100" />
        <ValorSensibilidadRGB SensibilidadR="40" SensibilidadG="40" SensibilidadB="40" />
      </RGB>
      <HSV>
        <ValorColorHSVMin H="100" S="40" V="40" />
        <ValorColorHSVMax H="140" S="60" V="60" />
      </HSV>
      <HLS>
        <ValorColorHLSMin H="100" L="40" S="40" />
        <ValorColorHLSMax H="140" L="60" S="60" />
      </HLS>
    </Camara2>
  </Color>
  <BusquedaRegiones>
    <Modo Activo="1" Dinamico="0" />
    <Camara1>
      <ParametrosClusters TamanoMinimo="20" DistanciaEntreClusters="10" />
```

```
        <Region1 Ancho="40" Alto="60" />
        <Region2 Ancho="40" Alto="60" />
      </Camara1>
      <Camara2>
        <ParametrosClusters TamanoMinimo="20" DistanciaEntreClusters="10" />
        <Region1 Ancho="40" Alto="60" />
        <Region2 Ancho="40" Alto="60" />
      </Camara2>
    </BusquedaRegiones>
    <PosicionOrigen X="200.0" Y="200.0" Z="200.0" Angulo="45.0" />
    <PosicionCamaras>
      <Camara1 X="2000.0" Y="0.0" Z="1000.0" Angulo="45.0" />
      <Camara2 X="0.0" Y="2000.0" Z="1000.0" Angulo="45.0" />
    </PosicionCamaras>
    <ParametrosCamaras>
      <Comunes ValorPixel="0.013780" Ancho="320" Alto="240" />
      <Camara1 DistanciaFocal="3.850" />
      <Camara2 DistanciaFocal="3.850" />
    </ParametrosCamaras>
    <ParametrosConexion Enviar="0" EnviarRelativo="1" IP="127.0.0.1" Puerto="10000" />
    <ConfiguracionFichero Guardar="1" Debug="1" Nombre="fichero.txt" NumeroImagenes="10000"
Separador="," />
</Configuracion>
```

Los descripción de los distintos parámetros y sus valores posibles es la siguiente:

- FicheroConfig:
  - Nombre: nombre del fichero de configuración.
- ModoEjecución:
  - MostrarVentanas: Indica si se muestran las ventanas con las imágenes durante la ejecución de la aplicación. Valores posibles:
    - 0 = No se muestran las imágenes.
    - 1 = Se muestran las imágenes.
- Color:
  - EspacioColor: Espacio de color que se utilizará para el sistema de localización. Valores posibles:
    - 0 = RGB
    - 1 = HSV
    - 2 = HLS
  - Camara1 y Camara2:
    - RGB:
      - ValorColorRGB: Valor aproximado del color en el espacio RGB que se utiliza para el sistema. Valores posibles: 0..255
      - ValorSensibilidadRGB: Valor de la ventana para la cual se aceptarán los colores de la imagen. Valores posibles: 0..255
    - HSV:

        - ValorColorHSVMin y ValorColorHSVMax: Valores mínimos y máximos para los parámetros del espacio HSV entre los cuales se aceptará el color. Valores posibles:
            - H: 0..360
            - S y V: 0..100
    - HLS:
        - ValorColorHLSMin y ValorColorHLSMax: Valores mínimos y máximos para los parámetros del espacio HSV entre los cuales se aceptará el color. Valores posibles:
            - H: 0..360
            - L y S: 0..100
- BusquedaRegiones:
  - Modo:
    - Activo: Indica si se utiliza el modo de búsqueda de regiones. Valores posibles:
        - 0 = Se utiliza el modo de búsqueda de regiones.
        - 1 = No se utiliza el modo de búsqueda de regiones.
    - Dinámico: Indica si se utiliza el modo de búsqueda de regiones con tamaño dinámico. Valores posibles:
        - 0 = Se utiliza el modo de búsqueda de regiones de tamaño fijo.
        - 1 = Se utiliza el modo de búsqueda de regiones de tamaño dinámico.
  - Camara1 y Camara2:
    - ParametrosClusters:
        - TamanoMinimo: Entero que indica el área mínima para que se considere un cluster. Algo que se procese de menos tamaño no se toma como cluster.
        - DistanciaEntreClusters: Entero que indica la distancia euclidiana mínima para que 2 clusters próximos se tomen como uno solo.
    - Region1 y Region2:
        - Ancho y Alto: Tamaño de un clúster utilizado cuando no está activo el modo dinámico.
- PosicionOrigen: Posición de origen del objeto a localizar respecto al origen en mm para envíar la posición relativa del objeto si se necesitase.
- PosicionCamaras: Posición y ángulo de inclinación de las cámaras respecto al origen en mm.
- ParametrosCamaras:
  - Comunes:
    - ValorPixel: Valor de un píxel en mm.
    - Ancho y Alto: Valores utilizados para los cálculos de calibración y localización.
  - Camara1 y Camara2:
    - DistanciaFocal: Distancia focal de cada una de las cámaras en mm.
- ParametrosConexion:
  - Enviar: Indica si se envían los datos resultantes a través de UDP. Valores posibles:

    - 0 = Se envían los datos a través de UDP.
    - 1 = No se envían los datos a través de UDP.
  - EnviarRelativo: Indica si se envían los datos respecto al punto de origen. Valores posibles:
    - 0 = Se envían los datos absolutos.
    - 1 = Se envían los datos relativos respecto a la posición de origen.
  - IP: IP de la máquina donde se envían los datos a través de UDP.
  - Puerto: Puerto de la máquina donde se envían los datos a través de UDP.
- ConfiguracionFichero:
  - Guardar: Indica si los datos resultantes se guardan en un fichero al finalizar la ejecución. Valores posibles:
    - 0 = No se guardan los datos en un fichero.
    - 1 = Se guardan los datos en un fichero.
  - Debug: Indica si se utiliza el modo debug para el fichero de salida (véase el apartado 4.2.7. para una descripción del formato del archivo). Valores posibles:
    - 0 = No se utiliza el modo debug.
    - 1 = Se utiliza el modo debug.
  - Nombre: Nombre del fichero donde se guardan los datos resultantes.
  - NumeroImagenes: Número total de datos que se guardan en el fichero.
  - Separador: Separador entre los datos en una misma medida.

4.2.5.1. Biblioteca

Para realizar el tratamiento de los xml hemos usado la biblioteca TinyXML. Esta biblioteca es libre y está desarrollada en C++. Además, es muy ligera y sencilla de utilizar, y aunque no ofrezca muchas funcionalidades es perfecta para nuestras necesidades, puesto que sólo necesitamos leer y guardar archivos xml.

4.2.6. Conexión mediante UDP

Para realizar la conexión con el ordenador encargado de tratar la información de la posición del cuatrimotor se utiliza el protocolo UDP.

El protocolo UDP es un protocolo del nivel de transporte basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. Tampoco tiene confirmación ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros; y tampoco se sabe si ha llegado correctamente, ya que no hay confirmación de entrega o recepción.

La ventaja de este protocolo frente a TCP es que es más veloz, ya que como se ha comentado no requiere de una conexión previa y no necesita asegurarse de que los datos se han enviado correctamente.

En resumen, utilizamos UDP ya que lo que nos importa es la velocidad frente a la fiabilidad. Además, el sistema está pensado para ejecutarse en un ordenador que esté conectado al que recibe los datos mediante un cable cruzado, por lo que va a ser fiable, ya que no se van a perder ni desordenar los datagramas en ningún momento.

La información que se envía en el datagrama son las coordenadas x, y, z de la posición del objeto y su ángulo de orientación. Esta información se envía en 4 bytes, utilizando 1 byte para cada dato.

La utilización de esta funcionalidad es opcional, y su uso se indica en el archivo de configuración mediante el parámetro *Enviar* de la etiqueta *ParametrosConexion*.

Las coordenadas y el ángulo que se envían pueden ser respecto al origen de coordenadas, o respecto a una posición inicial introducida en el archivo de configuración (*PosicionOrigen* del archivo de configuración). Esto último es utilizado para realizar el control de la posición en torno a un punto de equilibrio del objeto del que se realiza el seguimiento.

4.2.6.1. Biblioteca

Para el envío de datos mediante el protocolo UDP hemos utilizado la biblioteca Windows Sockets 2 que se puede encontrar dentro de las librerías de MinGW, ya que nos proporcionaba las funcionalidades que necesitábamos.

4.2.7. Fichero de salida

Esta funcionalidad se decidió incluir para dejar constancia de las distintas pruebas realizadas y después poder depurar la aplicación con mayor facilidad. Esta opción es configurable mediante el xml de configuración, en el que se indica si se quiere utilizar o no, si se quiere usar un modo debug para proporcionar más información, el nombre del archivo de salida, el número de medidas que se quieren guardar en el fichero y el separador que se utilizará para separar las distintas medidas.

En este archivo se guardan los siguientes datos por cada medida tomada dependiendo de si se encuentra activado o no el modo debug:

- Modo debug desactivado:
  tiempo, coordenada x, coordenada y, coordenada z, ángulo
- Modo debug activado:
  tiempo, píxel x imagen 1 cámara 1, píxel y imagen 1 cámara 1, píxel x imagen 2 cámara 1, píxel y imagen 2 cámara 1, píxel x imagen 1 cámara 2, píxel y imagen 1 cámara 2, píxel x imagen 2 cámara 2, píxel y imagen 2 cámara 2, coordenada x, coordenada y, coordenada z, ángulo, nuevo

El valor tiempo será el valor en milisegundos desde el comienzo de la ejecución en que se ha tomado la medida, las coordenadas de los píxeles indican la posición de los objetos en cada una de las capturas realizadas por las cámaras, las coordenadas y el ángulo indican las posición y orientación del objeto en ese instante de tiempo, y el valor de nuevo será 0 si el valor ha sido copiado de la anterior medida realizada porque en el instante de tiempo no se disponía de toda la información necesaria, y será 1 si las coordenadas y el ángulo calculado han sido calculados de nuevo.

Gracias a este fichero de salida la tarea de depuración de la aplicación es mucho más sencilla, pudiendo comprobar los datos resultantes después de la ejecución, y dando la posibilidad de realizar gráficos con los datos obtenidos fácilmente con distintas herramientas como Matlab o Microsoft Excel.

Este es un fragmento de un fichero de salida en modo debug:

```
625,130,65,123,103,159,161,251,161,128.18,177.132,1.73554,89.8276,1
688,130,67,123,103,159,160,252,162,128.306,176.808,1.3235,89.3866,1
750,130,67,123,103,159,160,251,161,128.426,176.967,1.38353,89.641,1
844,130,66,124,103,160,160,251,161,128.574,176.731,1.62841,90.328,1
906,130,67,124,102,159,160,251,161,128.584,177.054,1.4998,89.9848,1
969,130,65,124,103,160,160,251,160,128.6,176.715,2.02148,90.7186,1
1031,130,66,124,102,160,161,251,161,128.41,176.895,1.67792,90.3659,1
1109,130,66,124,103,160,161,252,161,128.468,176.549,1.57746,90.4857,1
1172,130,66,124,102,159,160,252,161,128.491,176.879,1.83249,90.1174,1
1234,130,65,124,103,160,161,252,161,128.39,176.605,1.82309,90.6518,1
```

Este es el mismo fragmento anterior pero con el modo debug desactivado:

```
625, 128.18,177.132,1.73554,89.8276
688, 128.306,176.808,1.3235,89.3866
750,128.426,176.967,1.38353,89.641
844, 128.574,176.731,1.62841,90.328
906, 128.584,177.054,1.4998,89.9848
969, 128.6,176.715,2.02148,90.7186
1031,128.41,176.895,1.67792,90.3659
1109,128.468,176.549,1.57746,90.4857
1172,128.491,176.879,1.83249,90.1174
1234,128.39,176.605,1.82309,90.6518
```

4.2.8. Receptor UDP

El receptor UDP es una aplicación externa al sistema de visión diseñada para la recepción de datos que envía el sistema de localización para comprobar que estos son correctos y dibujarlos en un sistema de coordenadas para observar

gráficamente el movimiento que ha realizado el aparato. Además de permitirnos guardar los datos recibidos en un fichero de texto para su posterior estudio.

Esta aplicación se puede ejecutar tanto en el mismo computador en el que se ejecuta el sistema de medida como en otro computador de la red para comprobar que la velocidad de transmisión es correcta.

El receptor UDP está diseñado en Java. Para la visualización gráfica se ha usado la librería *processing* y para la recepción de los datos procedentes del sistema de localización a traces de UDP se han usado los sockets propios de Java.

El programa está dividido en tres partes, el interfaz gráfico cuyo comportamiento está definido en *JFrameCubo.java*; la parte de recepción a través de UDP encapsulada en la clase *RecepcionUDP.java* y la parte de dibujado de puntos y posicionamiento de la cámara de dibujado que se engloba e *CubicGrig.java*. Se ha usado una clase auxiliar *PV3D.java* para guardar las coordenadas que vamos recibiendo (X, Y, Z, Angulo).

Un manual de usuario del receptor UDP puede encontrarse en el Apéndice A.

Capítulo 5. Análisis y precisión del sistema de medida

5.1. Introducción

A continuación se describen diversas pruebas realizadas sobre el sistema para comprobar que realiza todas sus funcionalidades de forma correcta.

Se han detallado pruebas tanto de la calibración de la distancia focal y de la inclinación, como del sistema de localización, habiendo para esto último pruebas del cálculo de la posición y de la orientación a corta (en torno a 20cm) y media distancia (en torno a 1m).

5.2. Pruebas en el micromundo

5.2.1. Calibración de la distancia focal

La distancia focal es un parámetro intrínseco de la cámara que es necesario conocer para calcular la posición del objeto, puesto que es a esta distancia a la que se encuentra el foco, y todos los rayos de luz convergen en él.

Para las cámaras, habitualmente, el fabricante indica una distancia focal determinada; en el caso de las webcam empleadas para las pruebas la distancia focal indicada es 3.85mm, lo que hace pensar que la distancia focal obtenida por medio de la calibración debe ser próxima al valor indicado.

En el caso de la distancia focal hay que comentar que sólo se calibra en una ocasión, antes de utilizar las cámaras por primera vez, puesto que con la distancia focal calculada se harán los posteriores procesamientos, pero no cambiará de una ejecución a otra.

Para la calibración de la distancia focal se cuenta con dos objetos de referencia, además, es necesario conocer la posición de la cámara y las de los dos objetos para que, según se ha explicado en el Capítulo 2 en el apartado correspondiente a la calibración de la distancia focal, se pueda hallar el foco de la cámara como la intersección de las rectas formadas por los objetos y sus proyecciones en la imagen. De esta manera, habiendo obtenido el foco, la distancia focal es la distancia de este punto al plano de la imagen.

Las cámaras pueden estar en cualquier posición, siempre y cuando sea conocida, y es necesario que no estén inclinadas durante el proceso de la calibración, es decir, con inclinación 0°.

En la siguiente imagen se puede ver la colocación del entorno para esta prueba para la calibración de la distancia focal:

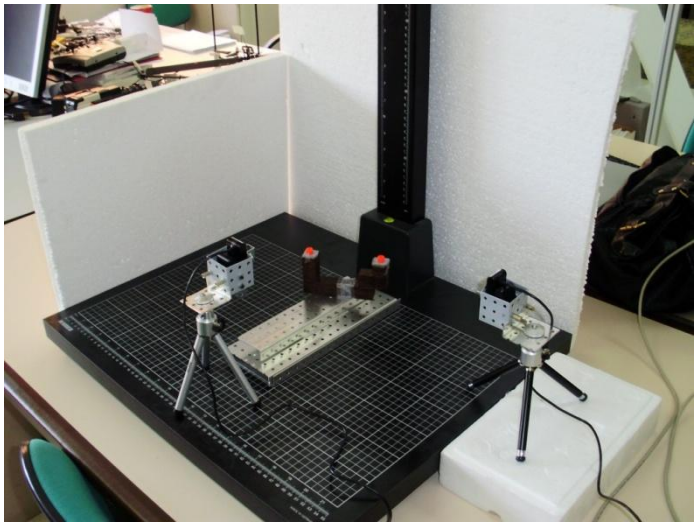


Figura 45: Entorno para la calibración de la distancia focal

Para calibrar, se colocan las cámaras en las posiciones deseadas y se incluyen en la llamada las posiciones de los objetos de referencia.

Las posiciones de las cámaras en la prueba realizada son:

CámaraXZ (200, 0, 100)
CámaraYZ (0, 250, 90)

Y las posiciones de los objetos:

Objeto de referencia 1 (200, 200, 65)
Objeto de referencia 2 (150, 290, 65)

Tras la calibración el resultado obtenido es:

Distancia focal cámaraXZ = 3.930583
Error en el cálculo de la distancia focal cámaraXZ = 0.218787

Distancia focal cámaraYZ = 3.715040
Error en el cálculo de la distancia focal cámaraYZ = 0.403784

Y los valores de las distancias focales se guardan en el archivo de configuración para poder trabajar con ellos en posteriores pruebas.

Como se ha explicado en el capítulo correspondiente a la calibración de la distancia focal, se produce un error en el cálculo que depende de la distancia entre las rectas que se cruzan, la desviación del foco estimado con respecto al eje óptico y la

distancia entre el foco estimado y el punto del eje óptico que se encuentra a una distancia igual a la focal de la cámara.

En este caso se puede observar que el error supone entre 5.5% y 10.8% de error con respecto al valor calculado o, lo que es lo mismo, por cada milímetro de distancia focal hay un error de entre 0.055mm y 0.108mm.

Además, la distancia focal indicada por el fabricante es de 3.85mm, lo que supone que entre las distancias calculadas y la indicada hay una diferencia aproximada de entre 0.08mm y 0.13mm.

5.2.2. Calibración de la inclinación

La inclinación es un parámetro extrínseco de la cámara que puede ser calibrado manualmente, pero la calibración de la inclinación implementada en el software desarrollado presenta un método de ayuda al usuario cómodo, seguro y fiable para estimar el ángulo de las cámaras.

En caso de querer estimar la inclinación por medio de esta calibración, es necesario que ya esté calculada la distancia focal o, en su defecto, emplear la indicada por el fabricante.

Para esta prueba se han utilizado los valores de la distancia focal obtenidos en la prueba anteriormente explicada.

Para realizar esta calibración sólo es necesario un objeto de referencia. Se colocan las cámaras inclinadas con el ángulo deseado, haciendo que el objeto de referencia se proyecte en el centro de la imagen y se indica en la llamada la posición del objeto de referencia. Debido a que para calibrar las dos cámaras a la vez, hay que hacer coincidir la proyección del objeto con el centro de ambas cámaras, resulta más cómodo calibrar la inclinación de una de las cámaras para, posteriormente, calibrar la otra.

Hay que destacar que es importante que las cámaras se encuentren bien colocadas, es decir, con sus ejes horizontales paralelos al suelo y centradas en sus respectivas posiciones introducidas en el archivo de configuración, ya que en caso contrario se podrían obtener deformaciones en las coordenadas calculadas, como se puede ver en las siguientes figuras.

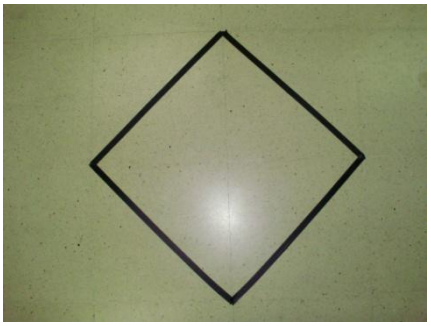


Figura 46: Figura real

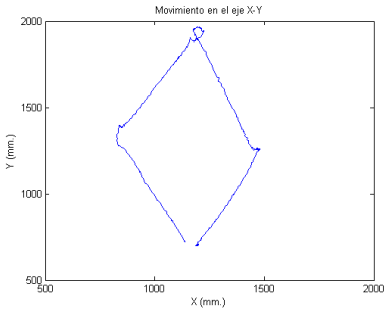


Figura 47: Figura obtenida

En las siguientes figuras se puede observar la colocación de ambas cámaras para la prueba expuesta a continuación:

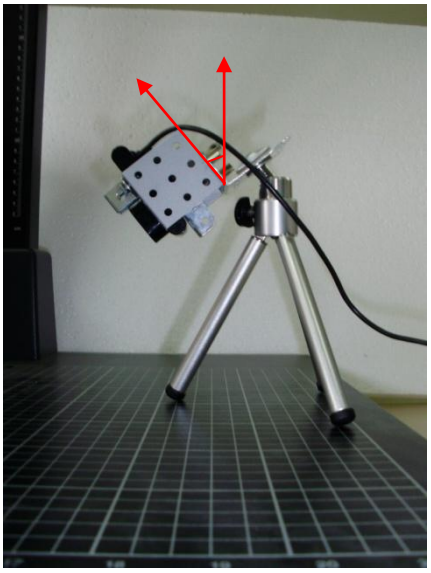


Figura 48: Inclinación de la cámaraXZ

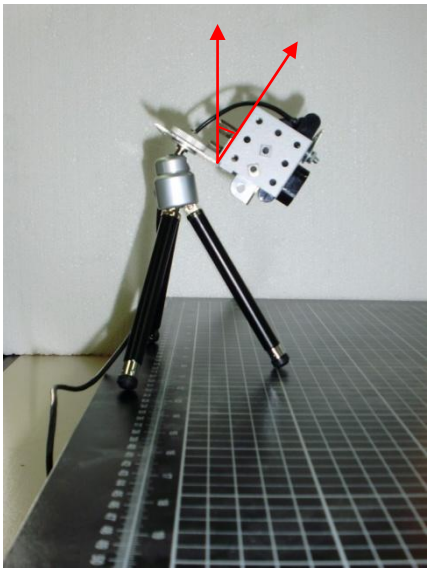


Figura 49: Inclinación de la cámaraYZ

Las posiciones de las cámaras para esta prueba son las siguientes:

CámaraXZ (150, 0, 95)
CámaraYZ (0, 200, 100)

Estando el objeto de referencia en la posición (150, 130, 5), para la cámaraXZ se ha obtenido el siguiente resultado:

Inclinación de la cámaraXZ = 35.18°

Y para la posición (165, 200, 5) del objeto de referencia, para la cámaraYZ se ha obtenido el siguiente resultado:

Inclinación de la cámaraYZ = 30.34°

Tras realizar los cálculos, los valores de las inclinaciones se guardan en el archivo de configuración para usarlos en posteriores ejecuciones del sistema.

Esta calibración permite la obtención del ángulo de inclinación de cada una de las cámaras, facilitando al usuario el uso del sistema y aportando una mayor certeza en la medición de los ángulos.

5.2.3. Cálculo de posición y orientación de un objeto sin movimiento

El sistema desarrollado calcula la posición y la orientación de cualquier objeto que se tome como referencia, parado o en movimiento.

En este caso, la prueba realizada es sin movimiento, es decir, cada captura de las cámaras devuelve la misma posición o posiciones contiguas debido a pequeñas variaciones al obtener el pixel central del objeto visualizado en la imagen.

En la siguiente figura se puede observar el entorno preparado para esta prueba:

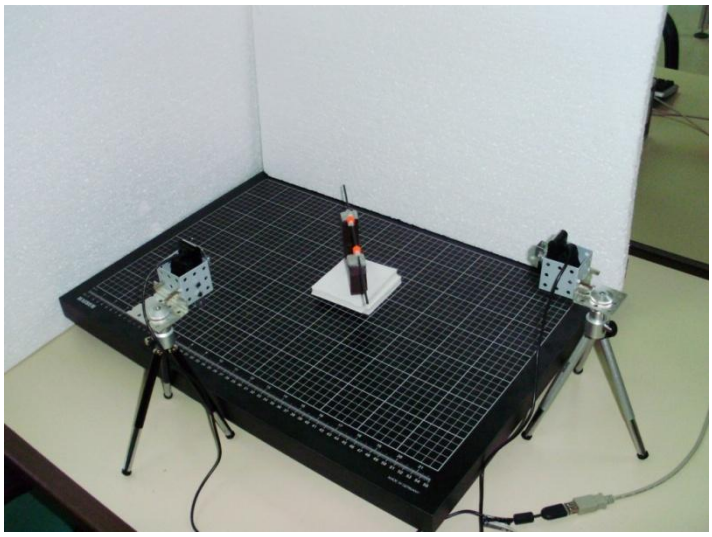


Figura 50: Entorno para la localización del objeto sin movimiento

Para esta prueba, únicamente es necesario colocar el objeto de referencia en la posición deseada y conocer las posiciones en las que están situadas ambas cámaras.

El objeto de referencia se encuentra en la posición (200, 250, 85) con una orientación de 45° ó 135° (es el mismo ángulo, depende del sentido que “considere” el sistema)

Y las cámaras en:

CámaraXZ (200, 0, 130) Ángulo 0°

CámaraYZ (0, 300, 130) Ángulo 0°

Se guarda la salida en un archivo de texto, habiéndolo indicado en el archivo de configuración y se obtienen valores, durante la prueba, similares a:

Objeto de referencia = (200.463913, 250.73682, 86.962779) Ángulo 46.252534°

Las Figuras 51 y 52 muestran el comportamiento del sistema para la localización del objeto situado en una posición fija.

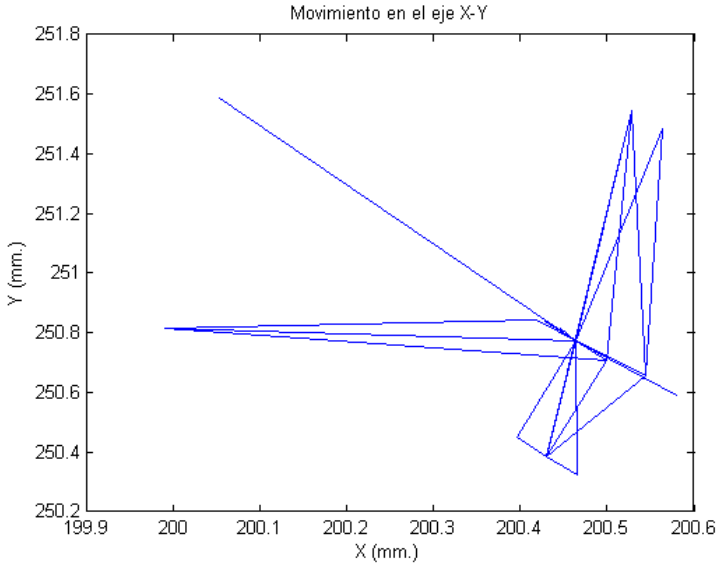


Figura 51: Posición del objeto de referencia durante la prueba

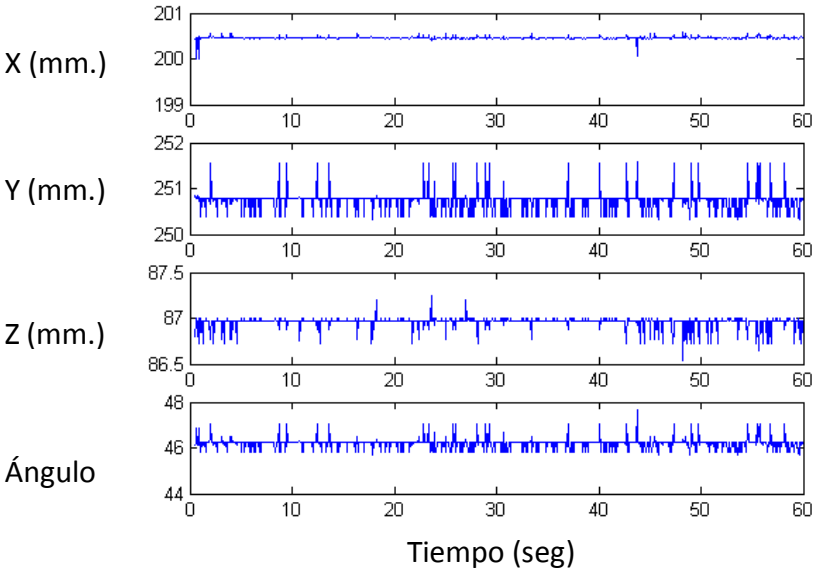


Figura 52: Variación de la posición y orientación del objeto de referencia

Como se puede observar en las figuras anteriores, las posiciones fluctúan entorno a unos valores muy próximos: la coordenada X calculada varía en el rango (200mm, 200.4mm), la coordenada Y en el rango (250.5mm, 251.5mm) y la coordenada Z en el rango (86.6mm, 87.2mm) y el ángulo en el rango (45.8°, 47°).

Estos rangos de variación suponen un error en el cálculo de alrededor de 0.4mm en la coordenada X, 1mm en la coordenada Y y 0.6mm en la coordenada Z. Estas variaciones se deben a que en el tratamiento de las imágenes previo al cálculo, se obtienen distintos píxeles, posiblemente contiguos unos de otros, de los objetos tomados como referencia.

La distancia de la cámaraXZ al objeto de referencia es de 25.4cm y la de la cámaraYZ es de 21.1cm. Y la distancia del punto obtenido con respecto a la posición real del objeto de referencia es de 2.14mm, por lo que se puede comprobar que el error supone entre 0.84% y 1.01% de la distancia a la que se encuentra el objeto, o lo que es lo mismo, por cada milímetro de distancia, el error es de entre 0.0084mm y 0.0101mm. El error cometido en el ángulo es de 1.252534°.

5.2.4. Cálculo con posición fija y orientación variable de un objeto

La orientación indica la dirección en la que se encuentran los dos objetos de referencia necesarios para esta prueba. Como se han empleado dos objetos del mismo color, sólo se calculará la dirección. En caso de querer conocer el sentido de los objetos, sería necesario que tuvieran colores diferentes para poder distinguirlos, pero esta funcionalidad está pensada para futuras fases del proyecto.

Para probar la variación de la orientación en caso de que la dirección del objeto de referencia cambie, se necesitan dos objetos de referencia colocados en posiciones cualesquiera y se gira sobre su propio eje de manera que sólo varíe el ángulo quedando la posición fija.

En este caso, se han girado los objetos 40°.

Se guarda la salida en un archivo de texto, habiéndolo indicado en el archivo de configuración, y se obtienen estos valores:

Situación inicial: (178.895, 223.286, 51.7666) 130.106°
Situación final: (182.214, 220.194, 50.5358) 91.1202°

La representación de la variación del ángulo de la prueba en función del tiempo se puede ver en la siguiente figura:

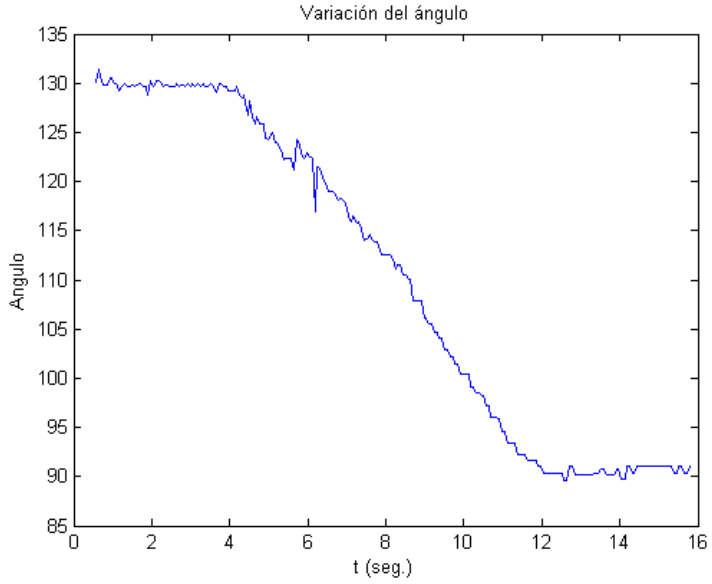


Figura 53: Gráfica de variación de la orientación

Como se puede observar en la Figura 53, la variación del ángulo es uniforme y casi lineal. La gráfica tiene un ruido medio de entre 0.5° y 1°, salvo alguna perturbación que no llega a superar los 7°.

5.2.5. Cálculo con posición variable y orientación fija del objeto

Para probar el comportamiento del localizador en caso de que la posición del objeto de referencia varíe pero no su orientación, se colocan los dos objetos de referencia en una posición determinada y se mueven hasta llegar a la posición deseada.

Para esta prueba se ha partido de una situación inicial, se ha movido el objeto de referencia 5cm. en el eje de las Y, 5cm. en el eje de las X y, posteriormente, se ha vuelto en diagonal a la posición inicial, siguiendo la forma de un triángulo rectángulo.

Habiendo guardado la salida en un archivo de texto, se han obtenido los siguientes valores:

Situación inicial:	(174.694, 329.556, 55.4995)	130.363°
Situación intermedia1:	(175.992, 280.209, 53.0093)	129.226°
Situación intermedia2:	(122.609, 279.794, 50.8223)	131.802°
Situación final:	(173.667, 328.657, 54.9501)	132.733°

La representación del cambio en la posición de la prueba se puede ver en las siguientes figuras:

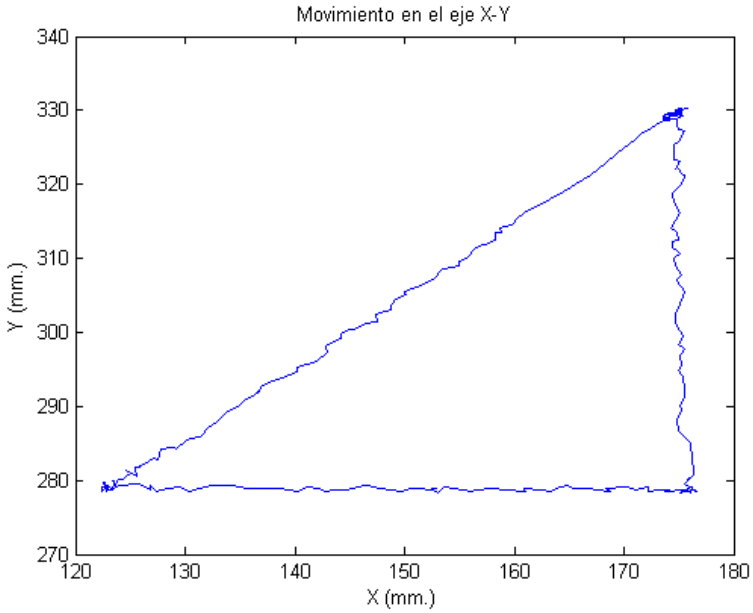


Figura 54: Distintas posiciones del objeto de referencia

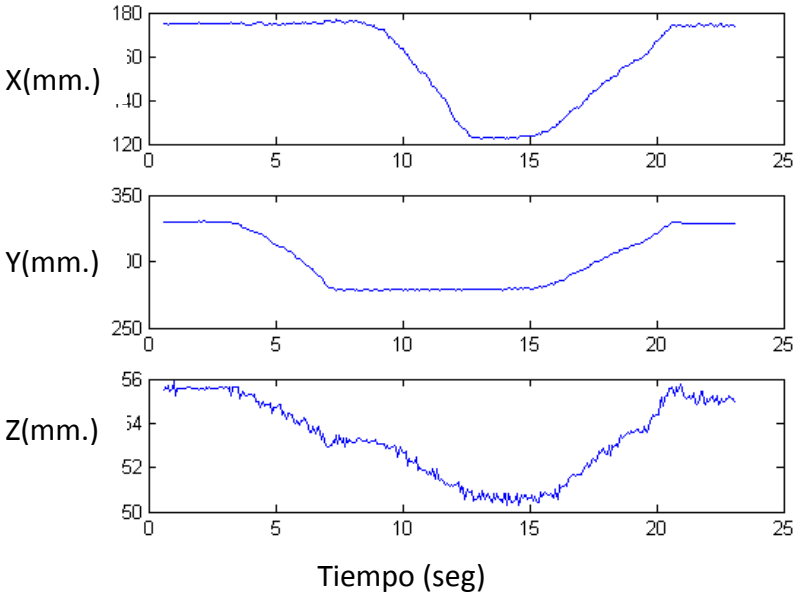


Figura 55: Coordenadas x, y, z del objeto durante el movimiento

Como se puede observar en estas últimas gráficas mostradas, la primera coordenada que varía es Y, para posteriormente variar X y más tarde ambas a la vez.

Por otra parte hay que destacar que, a pesar de no haber cambiado las alturas de los objetos de referencia durante la prueba, la Z varía en un rango menor de 6mm. Esto se debe a algunos parámetros de la cámara que, en próximas fases, sería interesante calibrar para poder subsanar.

5.3. Pruebas en el entorno de laboratorio

Una vez que se ha comprobado que la localización de un objeto parado o en movimiento a corta distancia ofrece unos buenos resultados, se han hecho unas pruebas en el entorno de laboratorio, a media distancia, para constatar que el funcionamiento no depende de la distancia a la que se encuentren las cámaras del objeto, siempre y cuando éste sea visible.

Para estas pruebas se ha seguido la trayectoria de figuras geométricas para poder ver posteriormente, mediante las gráficas obtenidas gracias a los datos extraídos de los ficheros de salida, si la localización realizada es correcta.

En primer lugar, se ha seguido la forma de un cuadrado sobre el suelo para comprobar el seguimiento de la posición colocando las cámaras a una distancia máxima de 1.2m de los objetos de referencia. El entorno se puede observar en la siguiente figura:

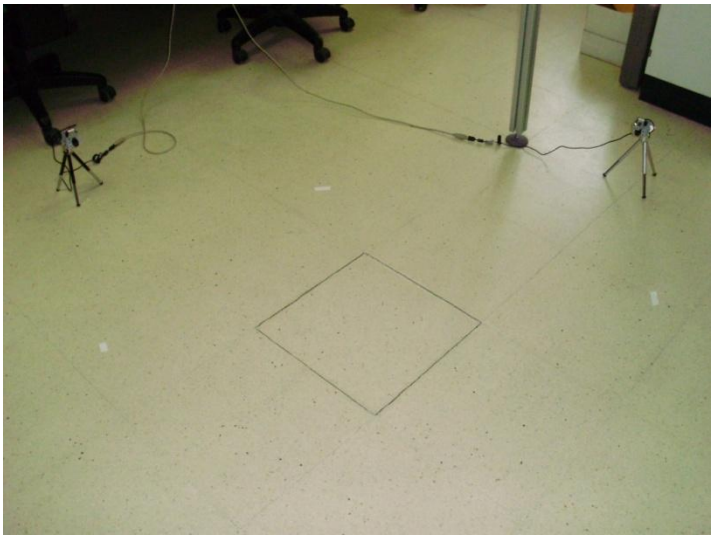


Figura 56: Cuadrado trazado para la localización

Las posiciones de las cámaras en el entorno son las siguientes:

CámaraXZ (1200, 0, 300) Ángulo 0°
CámaraYZ (0, 1200, 300) Ángulo 0°

Tras realizar la prueba, se han guardado los resultados en un archivo de texto y se han representado, obteniéndose los siguientes resultados:

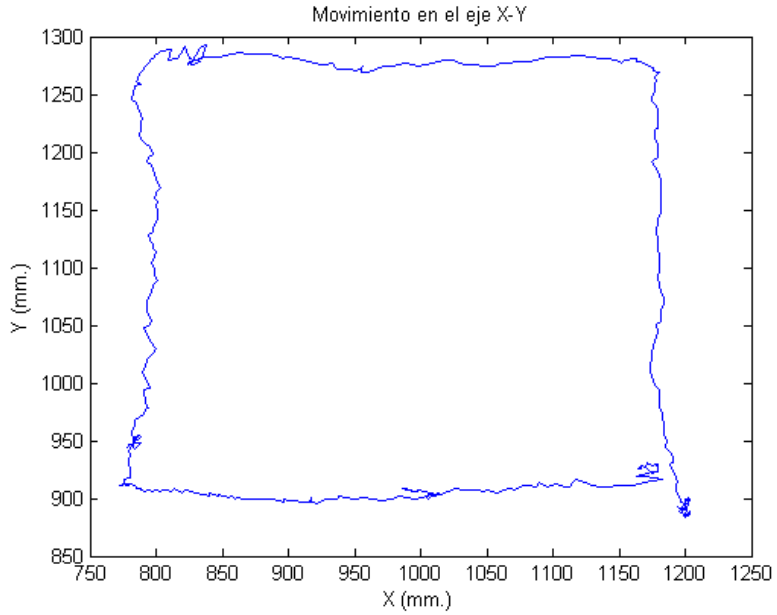


Figura 57: Localización del objeto de referencia siguiendo la trayectoria de un cuadrado

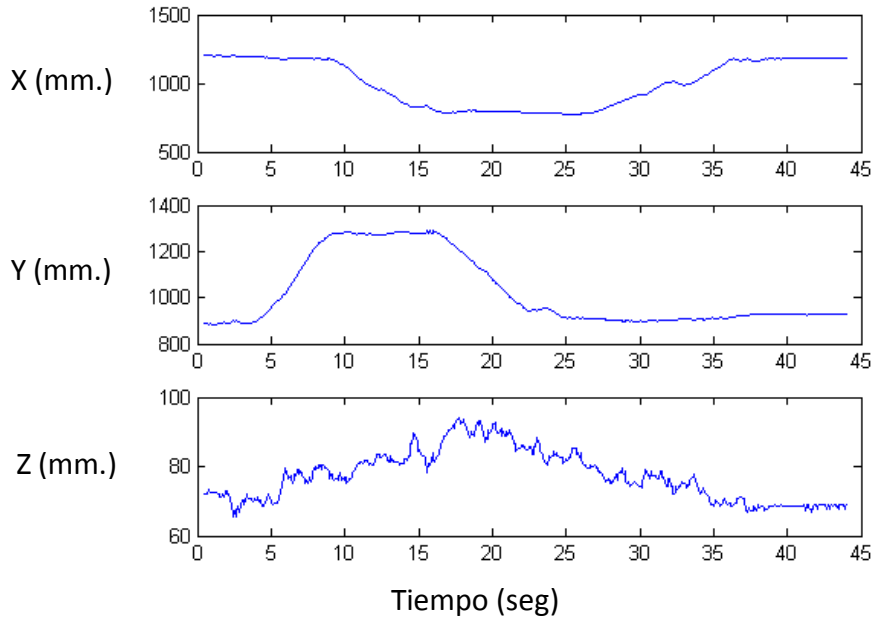


Figura 58: Coordenadas del objeto de referencia siguiendo la trayectoria del cuadrado

Esta misma prueba se ha realizado también siguiendo la trayectoria de un rombo en el suelo para observar, a parte de los cambios en la posición, la variación en el ángulo.

En este caso las cámaras se han colocado a mayor altura e inclinadas para evitar que los objetos de referencia se tapen entre sí en los giros y no se pueda calcular el ángulo en ese instante. El entorno se puede observar en la siguiente figura:

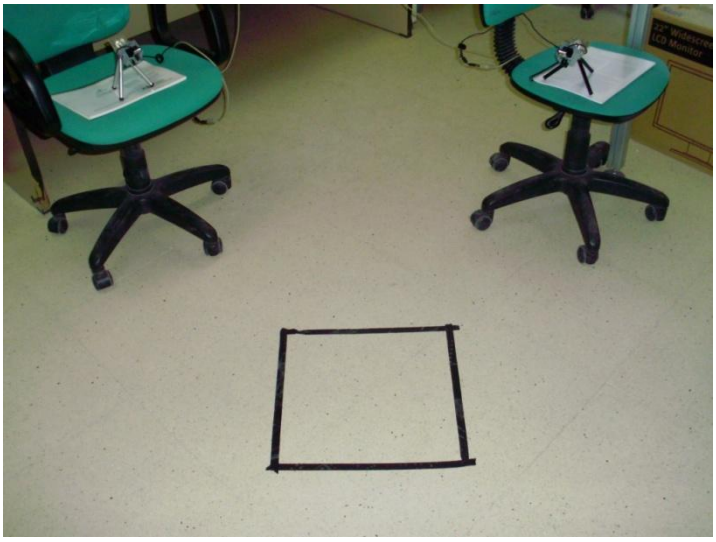


Figura 59: Rombo trazado para la localización

Las posiciones de las cámaras en el entorno son las siguientes:

CámaraXZ (800, 0, 600) Ángulo 40.38°
CámaraYZ (0, 800, 570) Ángulo 38.64°

Los resultados obtenidos mediante la representación del fichero de salida son los siguientes:

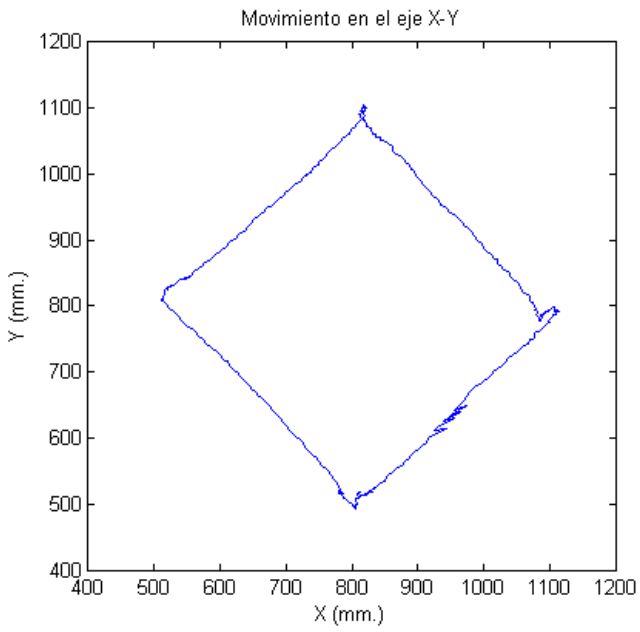


Figura 60: Localización del objeto de referencia siguiendo la trayectoria de un rombo

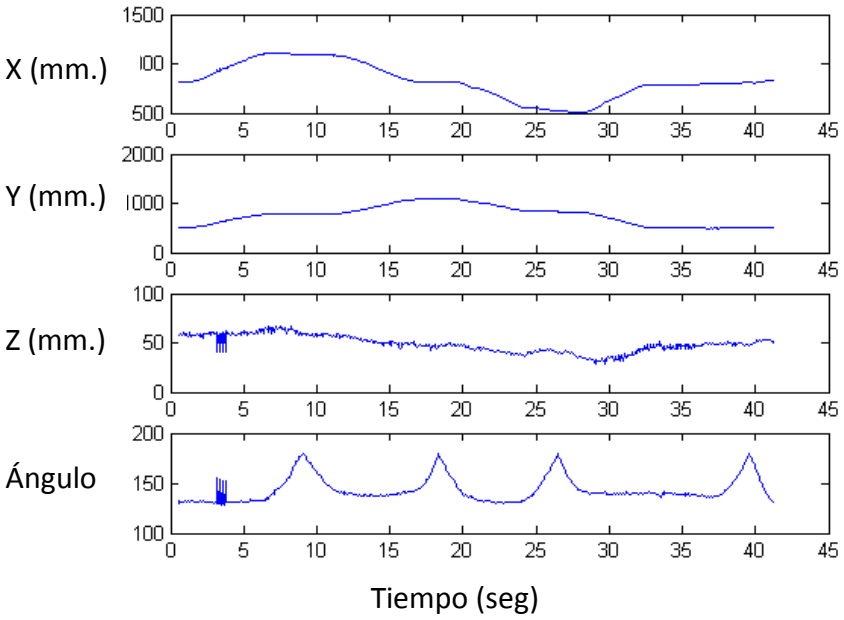


Figura 61: Coordenadas del objeto de referencia siguiendo la trayectoria del rombo

Como se puede observar, las gráficas 57 y 60 representan adecuadamente las figuras que se han seguido con los objetos de referencia, por lo que se puede concluir que el sistema realiza un correcto seguimiento de la posición a media distancia.

Además, en esta última gráfica se puede observar la variación que ha seguido el ángulo, obteniéndose correctamente un cambio progresivo de 90° en cada giro al realizar el cambio de lado.

Como en las pruebas a corta distancia se puede observar, se obtiene una variación en la coordenada Z a pesar de no haber variado la altura. Como se ha comentado anteriormente, sería interesante poder corregir este error en futuras versiones del sistema.

Capítulo 6. Ejemplos de uso

6.1. Introducción

A continuación se detallan posibles ejemplos de uso del sistema desarrollado. Estos ejemplos dan una idea del comportamiento del sistema en tareas de seguimiento de la trayectoria de móviles y del control de su posición.

6.2. Control de posición de un cuatrimotor

El sistema desarrollado es capaz de realizar el seguimiento de un móvil calculando, con cada captura que hagan las cámaras, las distintas posiciones por las que pasa el móvil en su trayectoria.

Con esta prueba se ha querido constatar el planteamiento inicial del desarrollo, que fue realizar el seguimiento de un cuatrimotor utilizado por un grupo de investigación de la Facultad de Físicas de la Universidad Complutense de Madrid.

Este es el cuatrimotor utilizado para esta prueba:

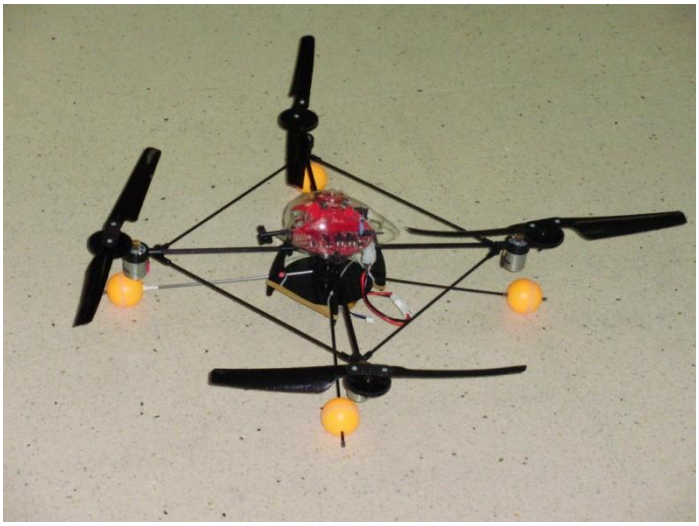


Figura 62: Cuatrimotor empleado en la prueba

Para realizar esta prueba, se colocan las cámaras en posiciones conocidas y se guardan estas posiciones en el fichero de configuración. Posteriormente se realiza la calibración de la inclinación, puesto que es necesario que las cámaras estén en alto e inclinadas hacia el suelo para que abarquen la mayor superficie posible y no pierdan los objetos de referencia debido a que se oculten unos a otros. Una vez realizada la calibración las posiciones de las cámaras son las siguientes:

CámaraXZ (1200, 0, 830) Ángulo 33.84°

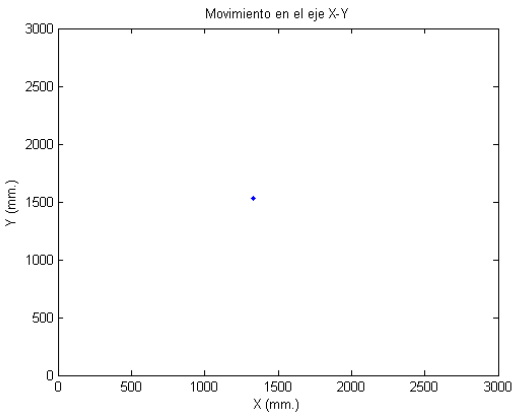
CámaraYZ (0, 1200, 960) Ángulo 36.07°

El entorno de prueba queda así:

Figura 63: Entorno de la prueba del seguimiento del cuatrimotor

A continuación se hace volar el cuatrimotor de forma controlada con una trayectoria aleatoria para que las cámaras vayan capturando las imágenes de la escena. Con el procesamiento de estas imágenes, el sistema calcula la posición del cuatrimotor durante su movimiento utilizando como objetos de referencia las 4 bolas que se encuentran debajo de los motores.

Al finalizar, se guarda la salida en un archivo de texto y, con la prueba realizada, se puede obtener el recorrido que ha realizado el cuatrimotor, del que se representa una parte a continuación:



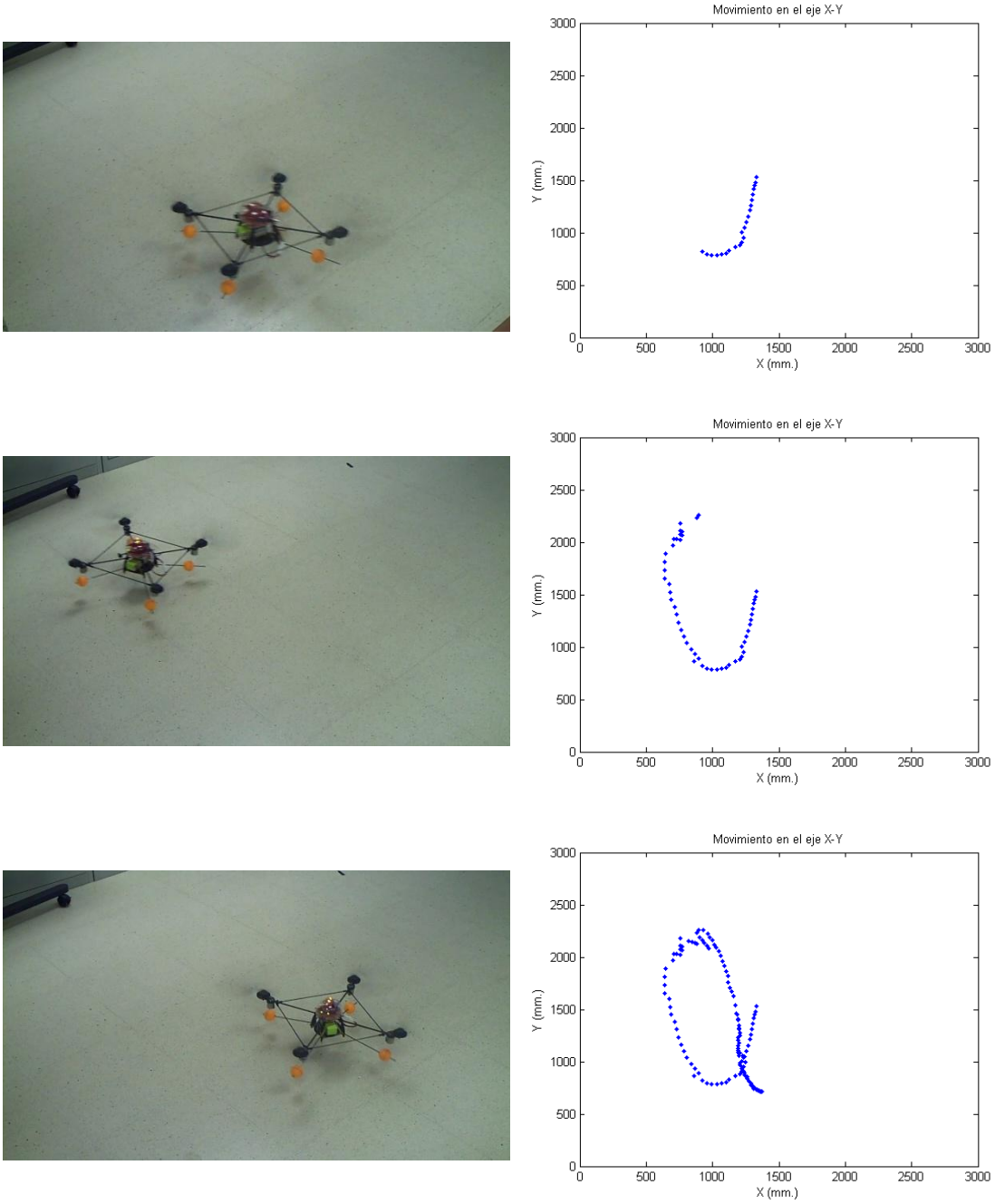


Figura 64: Trayectoria del cuatrimotor

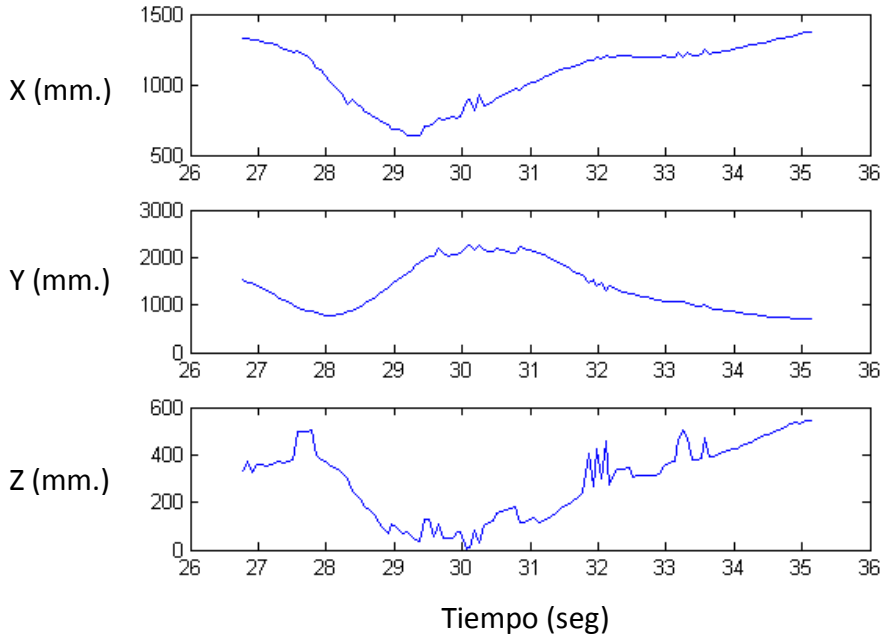


Figura 65: Coordenadas del cuatrimotor

Como se puede observar comparando la posición del cuatrimotor respecto a la pared de referencia que se encuentra al fondo de las imágenes de la Figura 64, el cuatrimotor realiza un movimiento elíptico variando su altura, que es localizado por el sistema y que se puede observar en las gráficas obtenidas como resultado en las Figuras 64 y 65.

6.3. Seguimiento de un móvil

En este caso, el seguimiento del móvil se realiza con un robot creado por alumnos de la asignatura de Robótica de la Facultad de Informática de la Universidad Complutense de Madrid. Este robot está dotado de un sensor de infrarrojos que permite que éste siga una trayectoria marcada como la que se muestra en la siguiente figura:

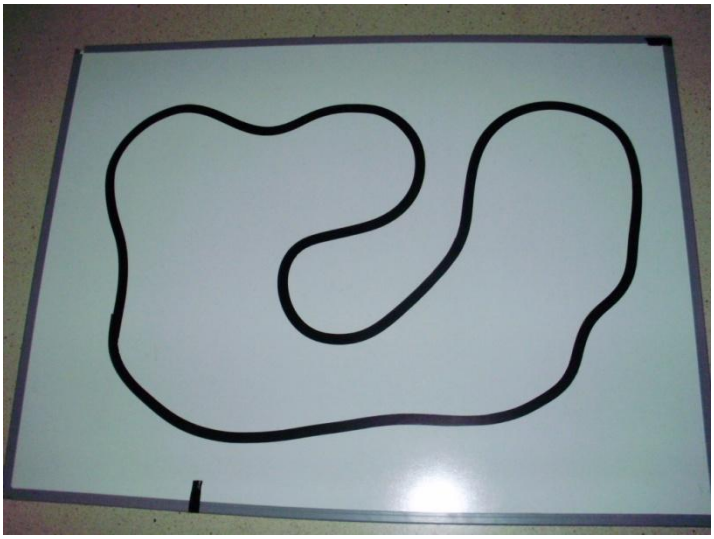


Figura 66: Trayectoria seguida por el robot

La diferencia con el seguimiento del cuatrimotor es que en este caso el robot se mueve por el suelo, variando sólo sus coordenadas x e y. Además, para esta prueba se ha utilizado como objeto de referencia el propio robot, ya que es de un color distinguible en el entorno y no es necesario añadirle otros objetos de referencia.

En este caso se sigue el mismo procedimiento que para la prueba del cuatrimotor. Se colocan las cámaras en las posiciones deseadas, indicando estas posiciones en el archivo de configuración y posteriormente se calibra su inclinación. Una vez finalizada la calibración, la posición de las cámaras en el archivo de configuración queda como se muestra a continuación:

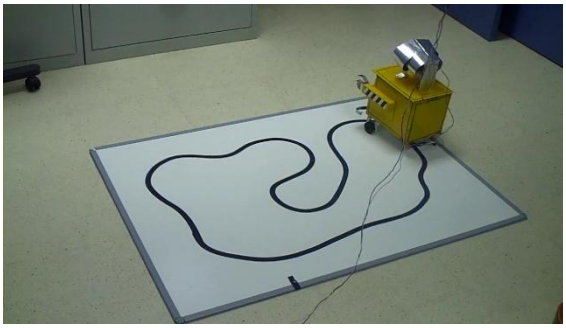
CámaraXZ (1600, 0, 850) Ángulo 25.356324°

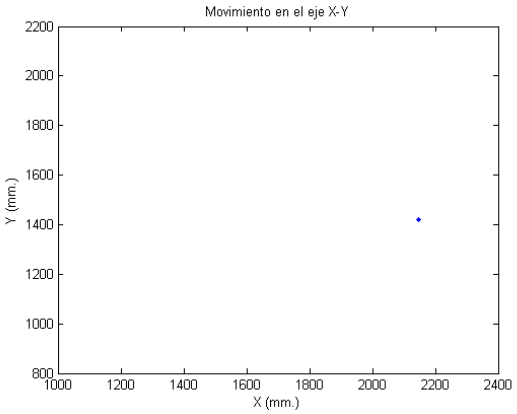
CámaraYZ (0, 1200, 950) Ángulo 20.722698°

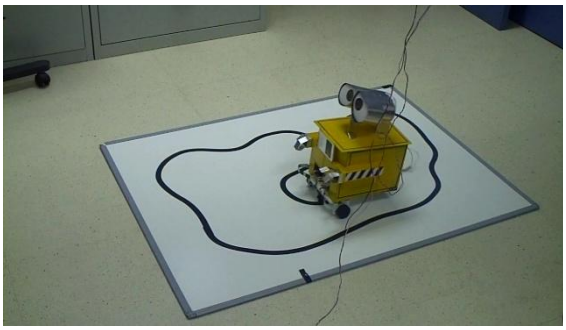
El panel donde está dibujada la trayectoria que seguirá el robot está colocado en un lugar completamente visible por las dos cámaras puesto que se necesita en todo momento que en las capturas de ambas cámaras aparezca el robot, quedando el entorno de pruebas así:

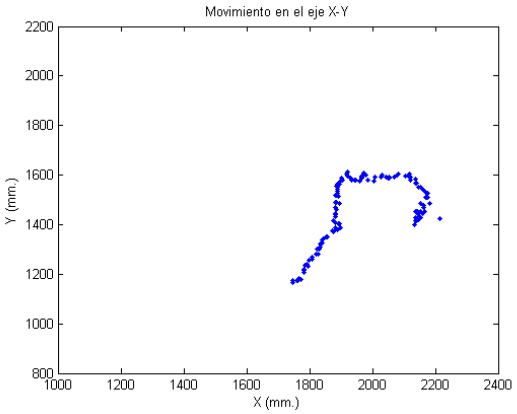
Figura 67: Entorno para la prueba del robot

Una vez que el sistema está preparado, se pone en funcionamiento el robot que, siguiendo la trayectoria marcada, vuelve a la posición inicial. Todo el recorrido queda reflejado en un archivo de texto del que se puede sacar la siguiente información acerca de las distintas posiciones que ocupa el robot durante el recorrido:









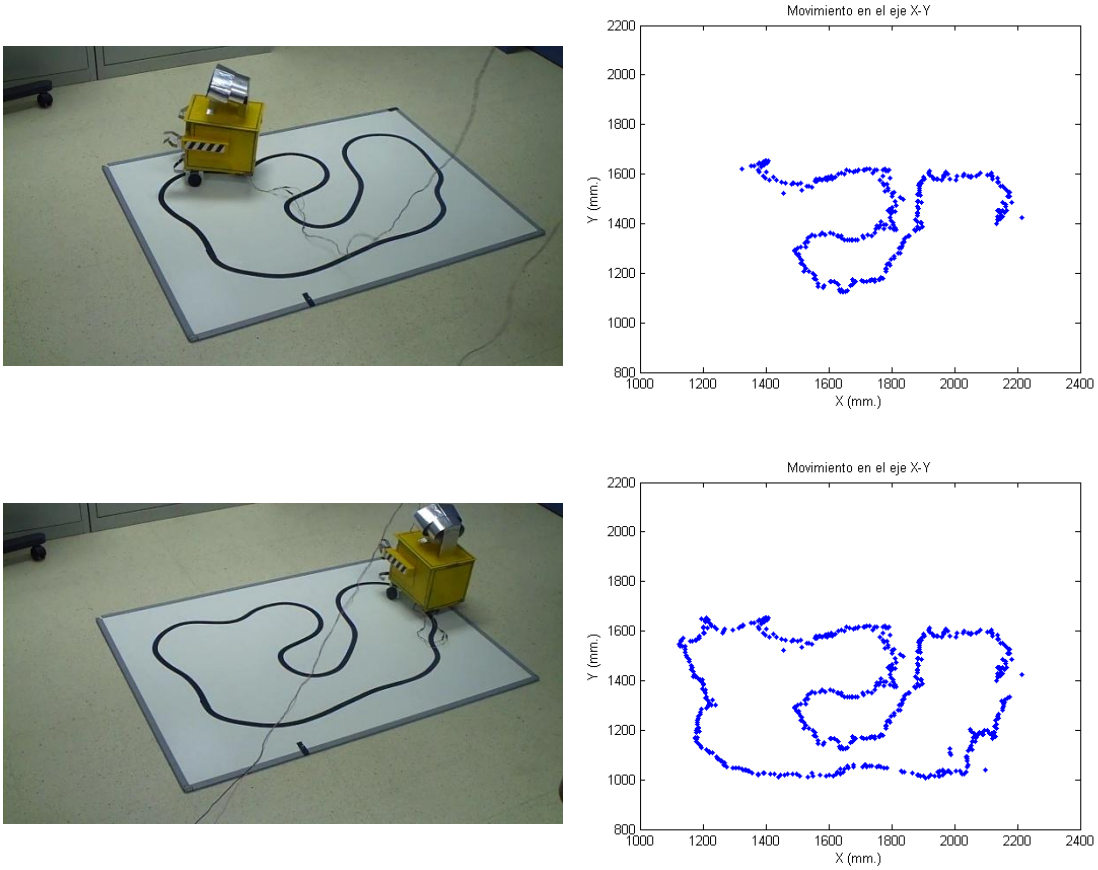


Figura 68: Trayectoria del robot

Como se puede observar en las anteriores figuras, la trayectoria calculada por el sistema es uniforme. Hay alguna posición calculada fuera del recorrido, pero por lo general éste es continuo. Si superponemos el recorrido original con el obtenido se puede observar que el resultado es muy similar.

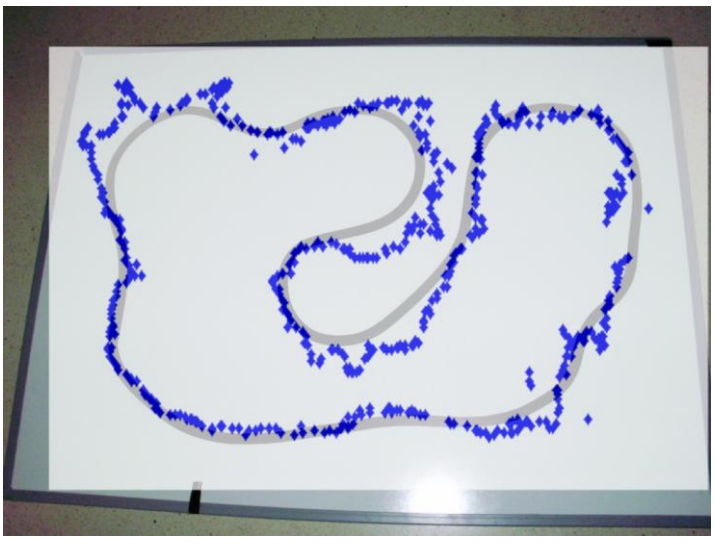


Figura 69: Comparativa de los recorridos

6.4. Localización y control de un mono-rotor

La idea inicial del proyecto, como ya se ha comentado, consiste en la localización de un cuatrimotor cuyas posiciones serán enviadas a un sistema de control que se encargará de manejarlo automáticamente utilizando esta información.

Hasta este momento, todas las pruebas que se han realizado han permitido confirmar que tanto la calibración como la localización se realizan satisfactoriamente. Sin embargo, una parte importante del proyecto consiste en el envío mediante UDP de las posiciones calculadas al sistema controlador del objeto considerado. En este caso, es ese aspecto del proyecto el que se pone a prueba.

Para esta prueba, se ha utilizado un mono-rotor, utilizado por el grupo de investigación ISCAR de la Universidad Complutense de Madrid, sujeto a un eje horizontal que se apoya sobre dos raíles guía verticales. Este mono-rotor se mueve a lo largo de los ejes verticales, por lo que hay que tener en cuenta que en la localización realizada sólo varía la posición en el eje Z real.

El sistema del mono-rotor original cuenta con una cámara en la base que visualiza una imagen situada en el eje horizontal, por lo que la imagen y el mono-rotor siempre estarán a la misma altura. La localización propia del sistema del mono-rotor se basa en la medida del área de la imagen visualizada por la cámara y el control consiste en, según la posición calculada, hacer subir o bajar el mono-rotor cambiando la velocidad de la hélice hasta conseguir una altura indicada por el usuario previamente.

Debido a que nuestro proyecto cuenta con un sistema de localización que puede utilizarse como sensor externo, para esta prueba se ha cambiado el sistema de localización original del mono-rotor por nuestro sistema. Para ello, se han utilizado dos marcas rojas similares a las empleadas en las demás pruebas a los lados del eje horizontal para que las cámaras distingan, en todo momento, dónde se encuentra el mono-rotor.

Para realizar esta prueba se colocan las cámaras en las posiciones deseadas y se guardan estas posiciones en el fichero de configuración. En este caso no es necesario que las cámaras estén inclinadas puesto que, como el movimiento del mono-rotor es vertical, con que las imágenes muestren en todo momento los ejes verticales de la estructura del mono-rotor es suficiente.

Para esta prueba se han colocado las cámaras en las siguientes posiciones:

CámaraXZ (550, 0, 210) Ángulo 0°

CámaraYZ (0, 800, 205) Ángulo 0°

El entorno de prueba queda así:

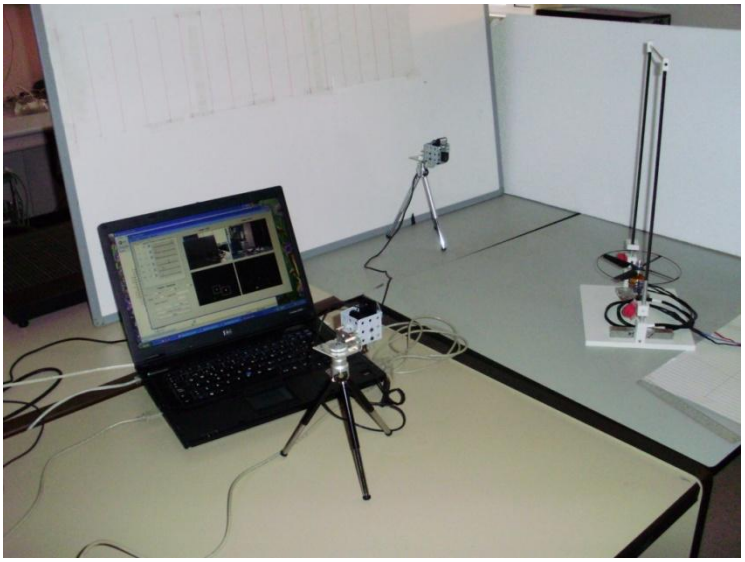


Figura 70: Entorno de la prueba de la localización y control de un mono-rotor

Una vez preparado el entorno, arrancado nuestro sistema de localización y estando el mono-rotor parado a altura 0mm, el usuario le indica al sistema la altura que quiere que el mono-rotor alcance. En este caso la altura deseada es 100mm.

El localizador, durante toda la ejecución, envía por UDP la posición del mono-rotor al controlador para que éste gestione el mono-rotor de la forma pertinente según las necesidades. Además, el controlador gestiona el mono-rotor a una frecuencia menor a la que recibe por UDP los datos.

Al iniciar la ejecución, como la posición del mono-rotor no es la indicada por el usuario sino que su altura es inferior, el controlador cambia la velocidad de la hélice del mono-rotor para que empiece a elevarse en la vertical de sus ejes. Cuando llega a la altura indicada por el usuario, el controlador vuelve a cambiar la velocidad de la hélice para que se mantenga la posición en la que está. Pasados unos segundos, el usuario indica otra altura, en este caso 0mm, lo que hace que se repita el proceso, esta vez haciendo que el mono-rotor descienda hasta que llega a la posición deseada.

Al finalizar la prueba, se guarda la salida en un archivo de texto que queda representada en las siguientes gráficas donde está representada la altura calculada, o coordenada Z del mono-rotor, en función del tiempo (en segundos).

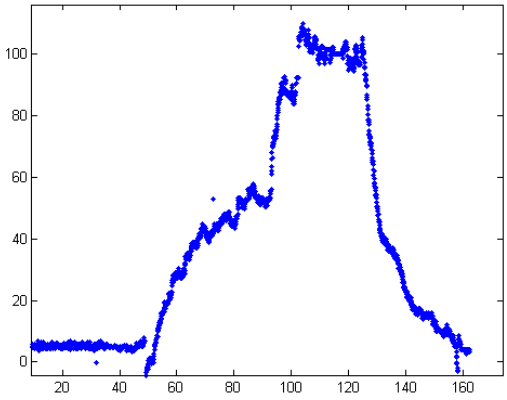


Figura 71: Salida del localizador (mm) a lo largo del tiempo

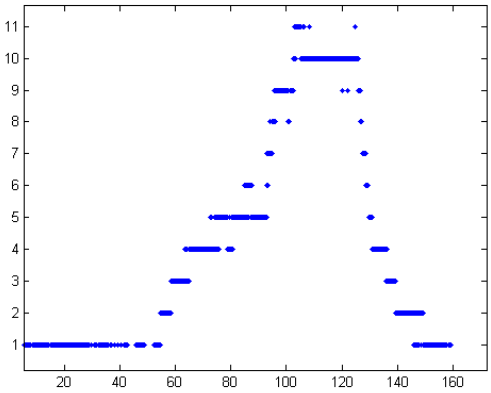


Figura 72: Datos enviados al controlador (cm) a lo largo del tiempo

Para adaptar los datos obtenidos del sistema de medida al sistema de control es necesario pasar los valores de mm a cm y truncarlos. Esta conversión se puede observar en las Figuras 71 y 72.

Además es importante la velocidad en el envío de datos por UDP puesto que, en caso de que el tiempo de transmisión fuera más lento que el tiempo en que el sistema de control del mono-rotor le envía las órdenes, cuando el controlador quisiera manejar el mono-rotor, el desfase produciría inconsistencias y movimientos innecesarios.

Como se ha observado en la prueba, la velocidad de envío de datos en suficiente, por lo que también queda comprobada la validez del sistema desarrollado como sistema de localización y envío de datos aptos para el control.

De este modo, se ha verificado que el localizador de visión sirve como sensor de posición para controlar un sistema, finalidad del proyecto, utilizando en este caso un sistema que ya existía para no perder el tiempo en el desarrollo del controlador ni de la gestión de la planta.

6.5. Control de posición de un helicóptero

El sistema está planteado para responder a la necesidad de control de un vehículo aéreo no tripulado localizado por el sistema de visión y posicionamiento. Para esta prueba se ha empleado el sistema de control desarrollado por el otro grupo de la asignatura de Sistemas Informáticos dirigido por Dr. José Jaime Ruz Ortiz y Dr. José Antonio López Orozco, como se propuso en el planteamiento de los proyectos a realizar.

Con esta prueba se pretende verificar que el sistema de visión desarrollado puede ser integrado con el controlador de posición del otro grupo de Sistemas Informáticos, manteniendo un helicóptero en una posición de equilibrio que será elegida por el usuario.

Para ello se han conectado por UDP los dos sistemas, nuestro sistema de visión y posicionamiento y el sistema de control del otro grupo. Mediante esta conexión, el sistema de control recibe la posición y la orientación relativas calculadas por nuestro sistema. Estas medidas se obtienen restando a la posición y la orientación absolutas las coordenadas del punto de equilibrio elegido anteriormente.

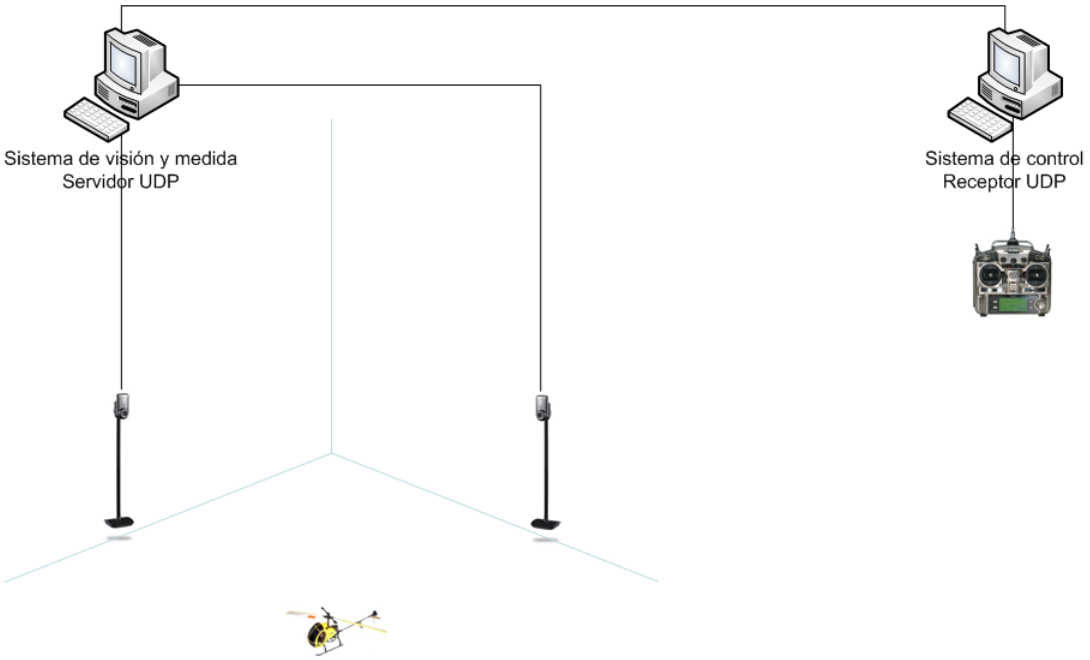


Figura 73: Esquema de realimentación

Una vez preparados los sistemas y el entorno (véase Figura 74), se realizan varias pruebas para ajustar la señal recibida por el controlador y los parámetros de los PID que utiliza. En este punto, el usuario indica la posición de equilibrio deseada y el sistema localizador empieza a calcular las distintas posiciones que adquiere el helicóptero, enviando por UDP las medidas halladas al sistema de control.

Con estas medidas, el controlador envía las señales necesarias al helicóptero para mantenerlo en la posición de equilibrio, haciendo variar la velocidad de sus hélices.

Los resultados obtenidos en esta prueba son prometedores: se realiza un control de la orientación del helicóptero que se puede percibir claramente porque al desplazarlo manualmente, retoma la orientación inicial; en cuanto a la posición en

altura se observa que al elevarlo disminuye la velocidad de la hélice y al bajarlo, aumenta.

Las pruebas iniciales, hasta que se tenga una seguridad grande de que el control del helicóptero se puede realizar sin problemas, se han realizado en un entorno controlado donde el helicóptero no puede desplazarse libremente (véase Figura 74), sino que gira sobre un eje fijo y se desplaza sólo 1-2 centímetros (en cada eje) de la base donde se encuentra.

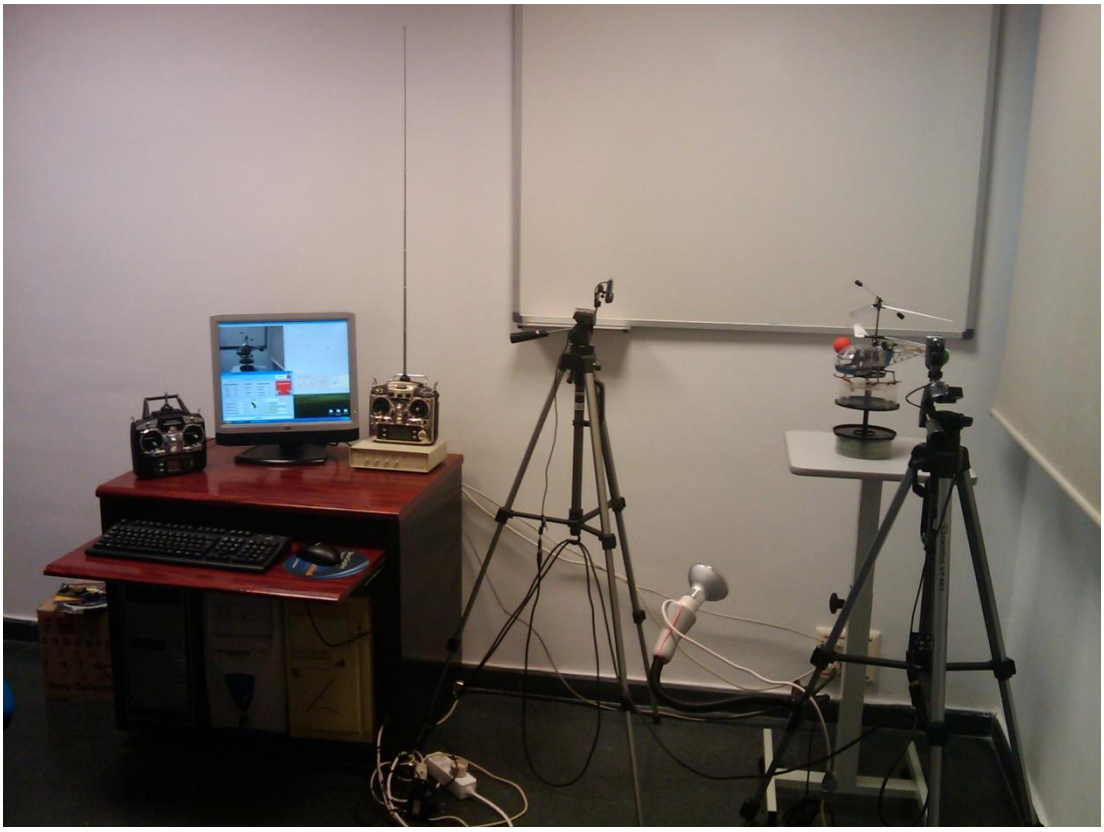


Figura 74: Entorno de laboratorio

Esta prueba, como se ha dicho, muestra que el sistema de localización desarrollado puede servir como referencia para obtener la orientación y posición del helicóptero. Pero todavía se necesitan realizar una serie de pruebas y comprobaciones antes de ponerlo en vuelo, ya que cualquier error llevaría a que se estrellase y, por tanto, al deterioro del vehículo.

Las verificaciones necesarias consisten en comprobar la respuesta del controlador ante variaciones bruscas, y su sintonización en caso necesario, así como verificar que en vuelo podrá controlarlo adecuadamente, puesto que de momento se ha realizado un control de posición.

Por otro lado, se debe verificar la robustez del sistema de visión, ya que una pérdida de la estima del helicóptero podría ser fatal. Para ello, se debe comprobar también la respuesta del controlador ante fallos en la medida de visión.

Capítulo 7. Conclusiones y trabajo futuro

7.1. Conclusiones

Inicialmente se planteó este proyecto como un sistema de visión para el posicionamiento de un cuatrimotor, pero se vio que con un poco de esfuerzo se podía generalizar esta idea y diseñar un sistema que fuera capaz de realizar el seguimiento de cualquier móvil, siempre que este contase con marcas u objetos de referencia, haciéndolo distinguible en el entorno para así poder realizar dicho seguimiento.

Con este planteamiento inicial se diseñó el sistema que fuese capaz de filtrar las imágenes capturadas quedándose con el color que el usuario de la aplicación escogiera, pudiendo así utilizar objetos de referencia de cualquier color, dejando su elección al usuario del sistema. Cabe destacar que esta selección de color se puede hacer utilizando 3 modelos de color distintos, RGB, HSV y HLS.

Para que el sistema se pudiera utilizar para el seguimiento de cualquier móvil, se desarrolló un modo mediante el cual no importaba con cuántos objetos de referencia contase el móvil. En este caso no es posible calcular la orientación, pero sí la posición calculando la media de todas las coordenadas de los objetos de referencia.

Además, el sistema es totalmente configurable, pudiendo seleccionar, además del color que se quiere filtrar, tanto la posición de las cámaras como sus parámetros, dando opción a que el sistema pueda ser utilizado en cualquier entorno controlado. Estas configuraciones se pueden guardar fácilmente en un archivo xml para tener distintas configuraciones dependiendo de dónde, con qué cámaras y para qué color de objetos de referencia se quieren realizar las pruebas.

A medida que fue avanzando el desarrollo del proyecto se realizaron diversas mejoras en la localización en las imágenes de los objetos de referencia, apareciendo la figura de los clústeres, que permiten realizar la búsqueda más eficientemente y permiten realizar un filtro para que afecten lo menos posible los brillos o ruidos que puedan aparecer en las imágenes debido al movimiento del móvil del que se está realizando el seguimiento.

Con todo esto al final se consiguió un sistema que realiza correctamente el seguimiento en tiempo real de un móvil, proporcionando su posición y vector dirección en todo momento, y que es capaz de enviar los resultados mediante protocolo UDP a otro ordenador de la misma red.

Las pruebas realizadas sobre el sistema, que se han descrito más detalladamente en los Capítulos 5 y 6, se han realizado con una separación de entre 15 cm y 2 m de distancia entre las cámaras y el móvil, obteniéndose errores de alrededor de un 1%.

Para realizar estas pruebas se han utilizado dos cámaras del modelo Logitech Quickcam Pro for Notebooks de 1,3 MP, obteniéndose una velocidad cercana a los 15 fps, pero también se han realizado pruebas con otras cámaras con las que se ha llegado a obtener 20 fps. Esto indica que la velocidad del sistema depende de la rapidez con las que las cámaras le proporcionan los frames a tratar y no del procesamiento, por lo que el sistema podría ir más rápido dependiendo de las características de las cámaras con las que se utilizase.

Observando los resultados que se obtienen del sistema en cuanto a la velocidad de adquisición y al ruido de la medida, se puede concluir que el sistema desarrollado puede utilizarse para el control de la posición de un móvil.

7.2. Trabajo futuro y posibles mejoras

Durante todo el proyecto se ha trabajado para desarrollar un sistema que cumpliera con todos los objetivos que se fijaron inicialmente, pero durante este tiempo se han ido planteando funcionalidades que sería interesante implementar y no se han realizado porque estaban fuera de nuestro marco de trabajo.

En este apartado se sugieren estas posibles mejoras que se pueden aplicar en un futuro al sistema para mejorar sus funcionalidades:

- Este sistema, como ya se ha explicado anteriormente, es capaz de medir la dirección del móvil en el espacio si dispone de dos objetos de referencia, pero sería muy útil también poder determinar su sentido.
  Esto sería posible si los dos objetos de referencia fueran de colores diferentes, pero implicaría realizar el proceso del tratamiento de las imágenes y de la localización de los objetos de referencia por duplicado. Sería interesante implementar esta funcionalidad en futuras fases del proyecto de una manera eficiente para que la velocidad del sistema se viera lo menos afectada posible.
- Nuestro sistema es capaz de realizar la calibración de varios parámetros de las cámaras, tanto intrínsecos como extrínsecos, que fueron los que se consideraron más relevantes para el correcto funcionamiento del sistema. Aún así, sería útil desarrollar otros métodos para realizar la calibración de otros parámetros, como los distintos tipos de distorsión que se producen en las cámaras de tipo pin-hole, que aunque no son tan importantes como los ya

calibrados, pueden ayudar para subsanar pequeños errores de medida que se puedan dar en los cálculos del sistema.

- Relacionado con esto último, como se ha comentado en las conclusiones de las pruebas del Capítulo 5, existe un pequeño error en el cálculo de la coordenada Z. Este error no se considera importante como ya se ha explicado, pero sería interesante subsanarlo corrigiendo los errores que se puedan dar por la perspectiva.
- También sería interesante implementar que, al realizar el cambio de un modelo de color a otro en la aplicación, este transformase los valores que se están utilizando a los del nuevo modelo, ya que actualmente se cargan los valores del archivo de configuración. En el Apéndice D se exponen las ecuaciones necesarias para realizar este cambio de modelo de color.

Bibliografía

Bibliografía

[A. Gardel Vicente, 2004]
Tesis Doctoral de Alfredo Gardel Vicente
http://www.depeca.uah.es/personal/alfredo/phd/tesis.htm

[Ahmed et al., 1999] Ahmed, M., Hemayed, E., and Farag, A. (1999).
Neurocalibration: a neural network that can tell camera calibration parameters.
In IEEE International Conference on Computer Vision, pages 463–468.

[Batista et al., 1998] Batista, J., Araújo, H., and de Almeida, A. (1998).
Iterative multi-step explicit camera calibration.
In Proc. Sixth International Conference on Computer Vision ICCV98, volume 15, pages 709–714.

[Bradski and Kaehler, 2008] Bradski G. and Kaehler A. (2008).
Learning OpenCV. O'Reilly, 1st Edition.

[Diego Pérez de Diego, 2006]
Trabajo presentado al seminario "Diseño y construcción de microrrobots" organizado por la Universidad de Alcalá
http://alcabot.org/seminario2006/Trabajos/DiegoPerezDeDiego.pdf

[Faugeras, 1993] Faugeras, O. (1993).
Three-dimensional computer vision: a geometric viewpoint.
MIT Press.

[Hartley et al., 1999] Hartley, R., Hayman, E., de Agapito, L., and Reid, I.(1999).
Camara calibration and the search for infinity.
In The proceedings of the International Conference on Computer Vision.

[Heikkilä and Silvén, 1996] Heikkilä, J. and Silvén, O. (1996).
Calibration procedure for short focal length off-the-shelf ccd cameras.
In Proc. of The 13th International Conference on Pattern Recognition, pages 166–170.

[Hengnian Qi, Wei Wang, Liangzhong Jiang, Luqiao Fan, 2007]
Automatic Target Positioning of Bomb-disposal Robot Based on Binocular Stereo Vision
http://www.atlantis-press.com/php/paper-details.php?id=1497

[J. J. Rodrigo, L. Acosta, J.A. Méndez, S. Torres]
Un robot jugador de "Ping-Pong" de bajo coste
http://www.cea-ifac.es/actividades/jornadas/XXII/documentos/F_13_R.pdf

[J. R. Martínez-de Dios, A. Ollero, L. Merino, y B. C. Arrue, 2006]
Medición automática y monitorización de fuegos mediante cámaras visuales y de infrarrojos
http://grvc.us.es/publica/revistas/documentos/Inc.Forest.pdf

[Javier García Ocón, 2007]
Proyecto de fin de carrera de Javier García Ocón
http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20070619JavierGarciaOcon.pdf

[Josep Isern González, 2003]
Estudio experimental de métodos de calibración y autocalibración de cámaras
http://gias720.dis.ulpgc.es/Gias/josep/TesisJosep.pdf

[Lourakis and Deriche, 1999] Lourakis, M. I. A. and Deriche, R. (1999).
Camera self-calibration using the singular value decomposition of the fundamental matrix: From point correspondences to 3D measurements.
Technical Report RR-3748, INRIA.

[Luong et al., 1992] Luong, Q., Faugeras, O., and Maybank, S. (1992).
Camera self-calibration: Theory and experiments.
In Proc. European Conference on Computer Vision, pages 321–334.

[Maybank and Faugeras, 1992] Maybank, S. and Faugeras, O. (1992).
A theory of self-calibration of a moving camera.
International Journal of Computer Vision, 8(2):123–151.

[Muljowidodo K., Mochammad. A. Rasyid., Sapto Adi N., and Agus Budiyono]
Vision Based Distance Measurement System Using Single Laser Pointer Design for Underwater Vehicle
http://www.centrums-itb.org/papers/Paper2008/SE_Paper46_rashid_IJMS.pdf

[Pollefeys et al., 1999] Pollefeys, M., Koch, R., and Gool, L. V. (1999).
Selfcalibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters.
International Journal of Computer Vision, 32(1):7–25.

[Proyecto AUTOPÍA, CSIC]
http://www.iai.csic.es/autopia/projet1esp.html

[Tsai, 1987] Tsai, R. Y. (1987).
A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the shelf tv cameras and lenses.
IEEE Journal of Robotics and Automation, RA-3(4):323–344.

[Vicente Milanés, Enrique Onieva, Teresa de Pedro, Ricardo García, Javier Alonso, Joshué Pérez, Carlos González]
Conducción autónoma en circuito de difícil maniobrabilidad
http://www.cea-ifac.es/actividades/jornadas/XXIX/pdf/299.pdf

[Willson, 1994] Willson, R. G. (1994).
Modeling and Calibration of Automated Zoom Lenses.
PhD thesis, The Robotics Institute. Carnegie Mellon University, Pittsburgh. Pennsylvania 15213.

[Zhang, 1998] Zhang, Z. (1998).
A flexible new technique for camera calibration.
Technical Report MSR-TR-98-71, Microsoft Research, Microsoft Corporation, Redmond, WA 98052.

Apéndices

9.1. Apéndice A. Manuales de usuario

9.1.1. Manual de usuario para el sistema con interfaz OpenCV

9.1.1.1. Introducción

Esta aplicación proporciona todas las funcionalidades implementadas del sistema de medida con una interfaz simple desarrollada mediante los módulos que proporciona la librería OpenCV.

Las funcionalidades que proporciona la aplicación son las siguientes:

- Localización: Funcionalidad principal del sistema, que se encarga de realizar la localización del aparato que dispone de los objetos de referencia. Proporciona al usuario la posición y la orientación del aparato respecto a unas coordenadas globales. Ofrece la posibilidad de enviar los datos obtenidos a través de datagramas UDP a otra computadora de su misma red.
- Cálculo de la posición inicial: Permite establecer una posición origen para el objeto de referencia para poder enviar datos relativos a esta posición a través de UDP.
- Calibración de la distancia focal: Este modo de ejecución permite calibrar las distancias focales de las cámaras.
- Calibración de la inclinación: Este modo de ejecución permite calibrar la inclinación de las cámaras.

9.1.1.2. Ejecución de la aplicación

Para ejecutar este sistema es necesario arrancarlo mediante un acceso directo o mediante la consola de comandos de Windows introduciendo unos parámetros de entrada que dependerán del modo de ejecución que se vaya a utilizar. Los diferentes parámetros a introducir son los siguientes:

- Localización.
  <ejecutable>.exe 1 <archivo de configuración>.xml
- Cálculo de la posición inicial.
  <ejecutable>.exe 2 <archivo de configuración>.xml
- Calibración de la distancia focal
  <ejecutable>.exe 3 <archivo de configuración>.xml x1 y1 z1 x2 y2 z2
  Siendo:
  x1, y1, z1: Las coordenadas reales del punto 1 para la calibración.

x2, y2, z2: Las coordenadas reales del punto 2 para la calibración.

- Calibración de la inclinación.

  <ejecutable>.exe 4 <archivo de configuración>.xml

Para todas las funcionalidades es necesario que el archivo de configuración que se quiere cargar esté construido correctamente, ya que en caso contrario dará error al cargarlo. Las posibles opciones de configuración de la aplicación se explican en el Capítulo 4 (Configuración mediante XML).

Al comenzar la ejecución de cualquiera de las funcionalidades obtenemos 4 ventanas, dos de ellas muestran las imágenes originales que captan las cámaras y los otras dos las imágenes después de haber realizado el procesamiento. Además, estas dos últimas ventanas cuentan con varias barras que permiten elegir el color que se quiere filtrar y otras opciones para tratar los clústeres.

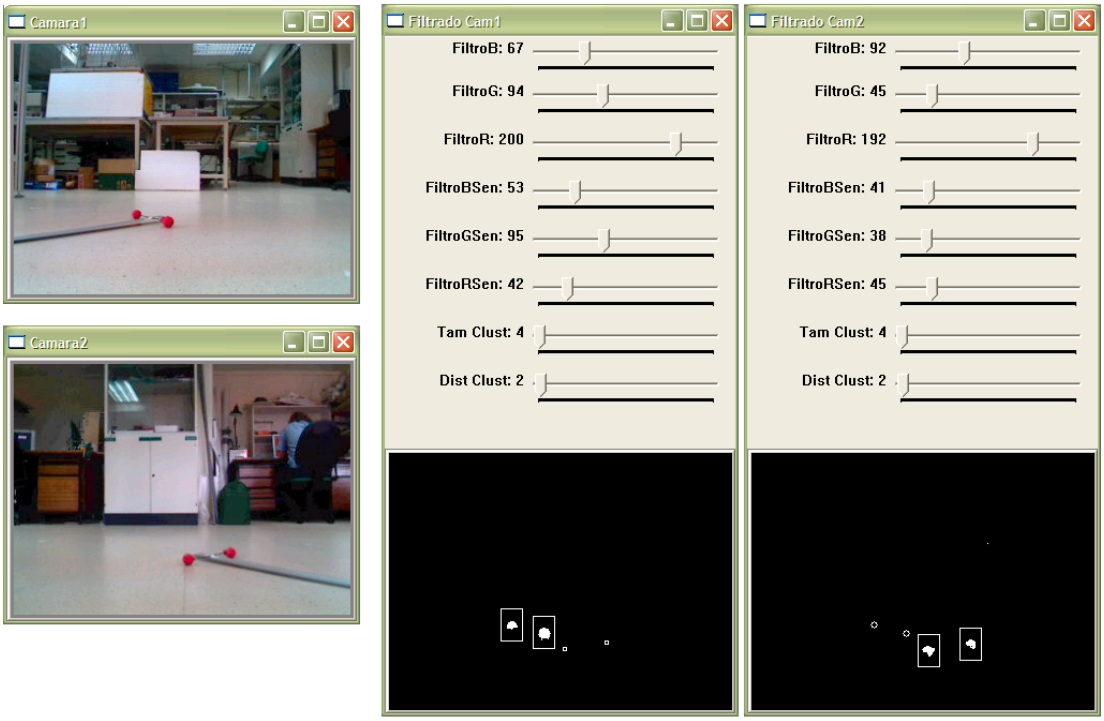


Figura 75: Interfaz OpenCV

A continuación describimos la ejecución de todas las funcionalidades de la aplicación.

Localización

Durante la ejecución de la aplicación obtenemos en pantalla los datos que se van obteniendo de la posición y el ángulo del objeto.

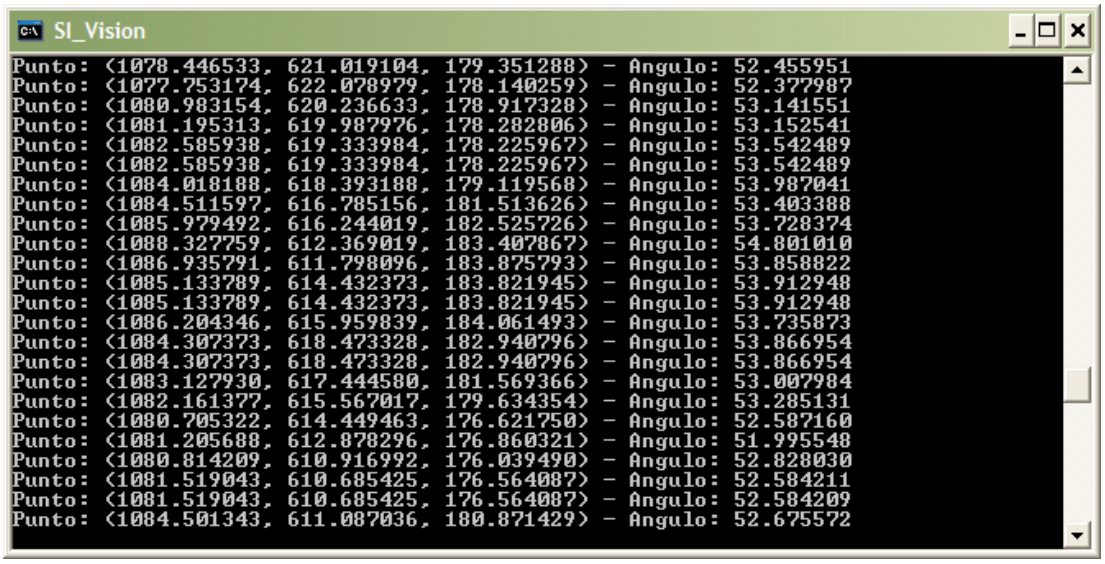


Figura 76: Ejecución de la localización

Podemos determinar la posición inicial del aparato presionando la tecla Enter.

Para detener la ejecución es necesario presionar la tecla Esc. Al detener la ejecución obtenemos los fps obtenidos durante la ejecución.

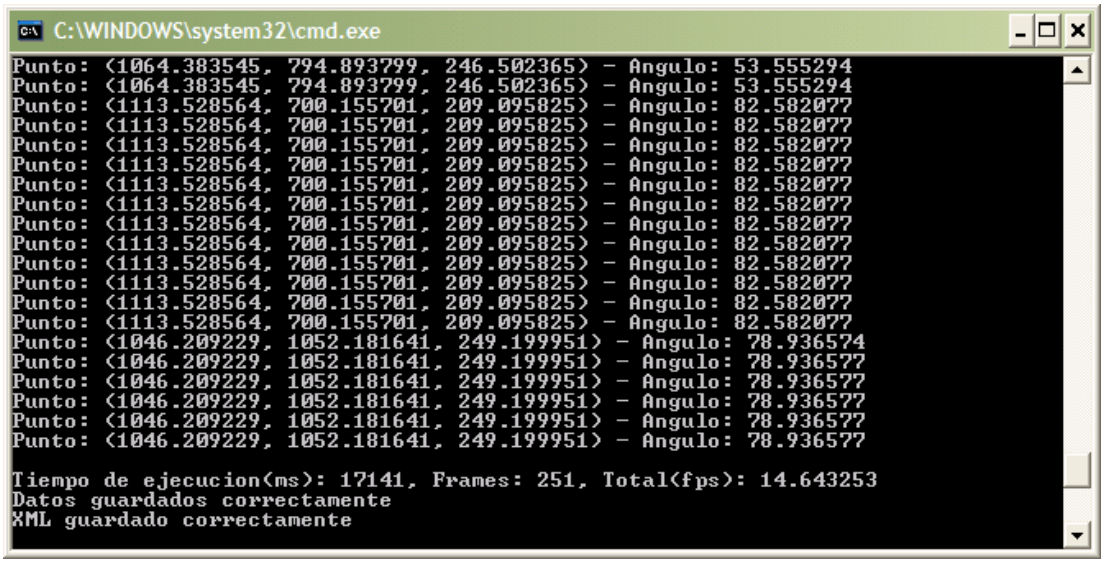


Figura 77: Fin de ejecución de la localización

Cálculo de la posición inicial

Durante la ejecución de la aplicación obtenemos en pantalla los datos que se van obteniendo de la posición y el ángulo del objeto.

Para detener la ejecución es necesario presionar la tecla Esc. Al detener la ejecución obtenemos por pantalla la posición y ángulo inicial y nos pregunta si queremos guardar estos datos. Si se guardan los datos finaliza la ejecución, en caso contrario

la aplicación peguntará si queremos continuar la ejecución para obtener otra vez los datos.

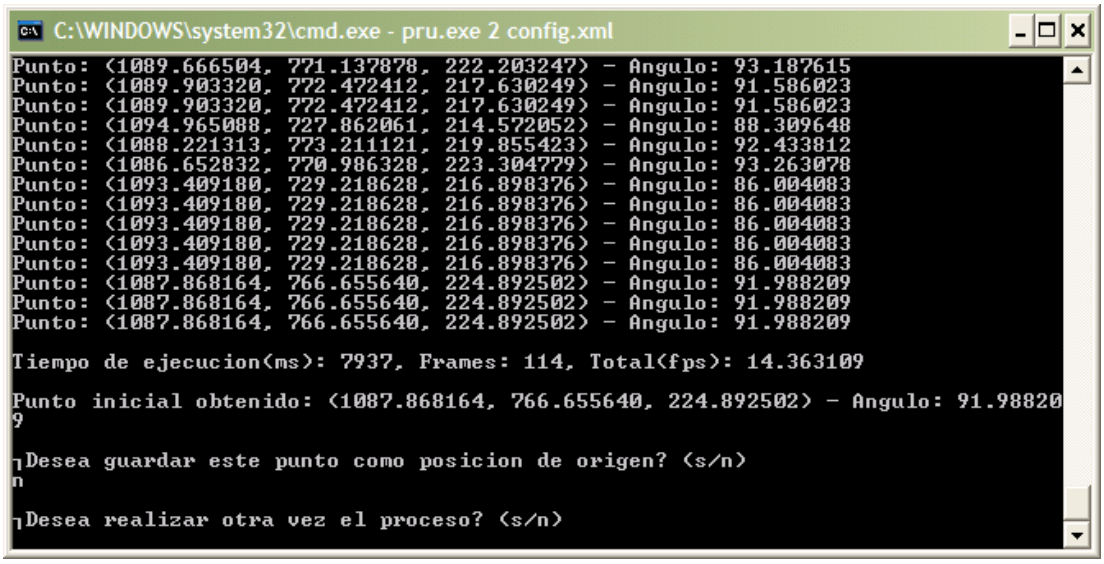


Figura 78: Fin de ejecución del cálculo de la posición inicial

Calibración de la distancia focal

Para ejecutar esta funcionalidad debemos haber introducido como parámetros en la llamada las coordenadas globales en mm de dos objetos de referencia del mismo color con los que se realizarán los cálculos.

Durante la ejecución de la aplicación obtenemos en pantalla los datos que se van obteniendo de la posición de los objetos de referencia en las imágenes para comprobar que se obtienen valores correctos.

Para detener la ejecución es necesario presionar la tecla Esc. Al detener la ejecución obtenemos por pantalla el valor de las distancias focales de cada una de las cámaras y el error obtenido en el cálculo. La aplicación pregunta si queremos guardar estos datos. Si se guardan los datos finaliza la ejecución, en caso contrario la aplicación peguntará si queremos continuar la ejecución para realizar otra vez el proceso.

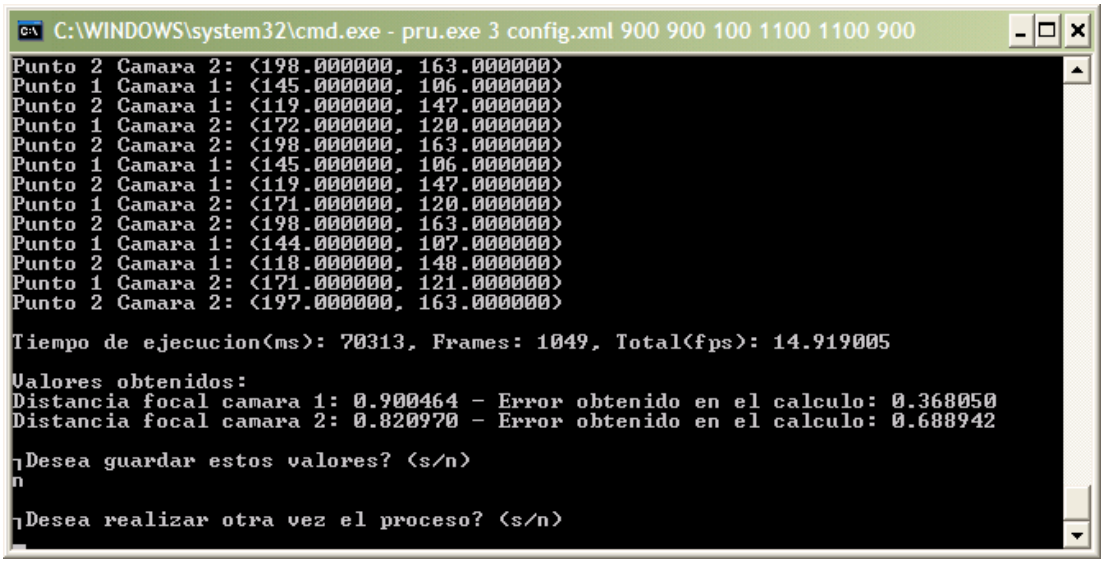


Figura 79: Fin de ejecución de la calibración de la distancia focal

Calibración de la inclinación

Esta funcionalidad consta de dos pasos, ya que las calibraciones de la inclinación de las cámaras no se pueden realizar a la vez. Por ello, primero se realizará la calibración de la cámaraXZ y después la de la cámaraYZ.

Durante la ejecución de la aplicación obtenemos en pantalla los datos que se van obteniendo de la posición del objeto de referencia en la imagen correspondiente para comprobar que se obtienen valores correctos. En la primera parte de la ejecución aparecerán los valores de la imagen de la cámara 1 y, tras haber realizado la calibración de esta cámara, los valores de la cámara 2.

Para que los cálculos se realicen correctamente se debe colocar un objeto de referencia en el centro de la imagen de la cámara con la que se esté realizando el proceso en ese momento. Tras colocarlo en el centro de la imagen se debe presionar la tecla Esc y nos pedirá introducir las coordenadas absolutas en las que se encuentra el objeto para poder realizar los cálculos. Después de esto obtendremos por pantalla el valor de la inclinación obtenido y nos preguntará si queremos continuar con la siguiente cámara o repetir el procedimiento con esta. Con la otra cámara el proceso será el mismo. Al finalizar el proceso con la cámara 2 se nos mostrará por pantalla los valores obtenidos con ambas cámaras y nos preguntará si queremos guardar los datos antes de finalizar la ejecución.

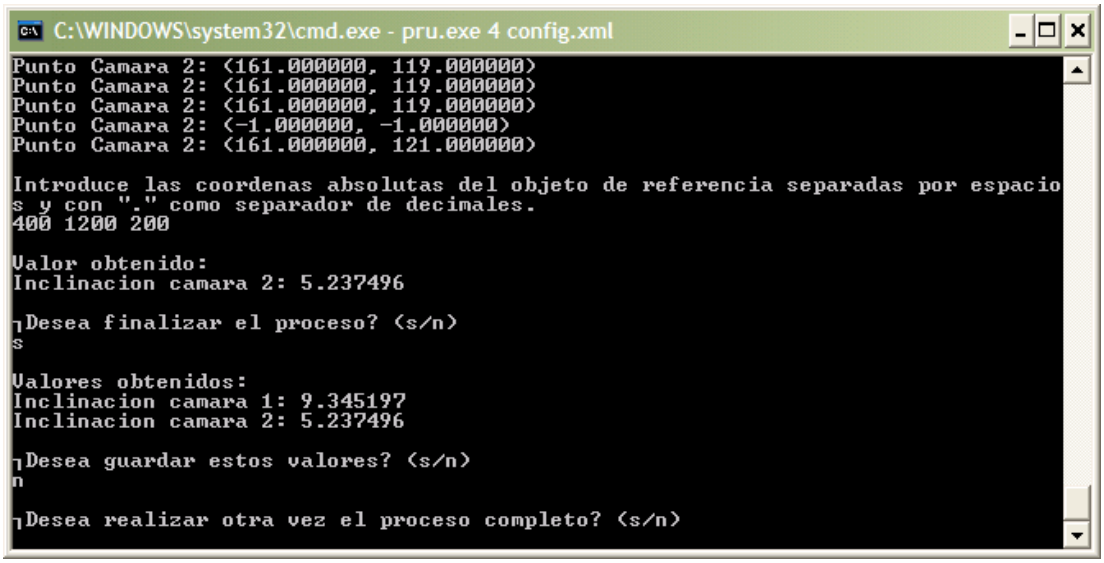


Figura 80: Fin de la ejecución de la calibración de la inclinación

9.1.2. Manual de usuario para el sistema con interfaz Visual Studio

9.1.2.1. Introducción

La aplicación de Visión diseñada con Visual Studio permite al usuario ejecutar el sistema de una forma completa y cómoda.

Está diseñado de manera que el usuario tenga toda la funcionalidad del sistema de una manera clara y sencilla.

Las funcionalidades que esta aplicación ofrece son:

- Localización: funcionalidad principal del programa, que se encarga de realizar la localización del aparato que dispone de los objetos de referencia. Proporciona al usuario la posición y la orientación del aparato respecto a unas coordenadas globales. Ofrece la posibilidad de enviar los datos obtenidos a través de datagramas UDP y de guardar dichos datos en un fichero de texto.

  En esta modalidad se permite establecer como posición de origen del objeto una que esté dando el modulo de localización pulsando un botón. Esta posición de referencia se usa para el envío de datos a través de UDP.

- Calibración: mediante este modo de ejecución se permite calibrar algunos parámetros de las cámaras. Está dividido en dos modos:
  - Calibración de la distancia focal: Permite calibrar la distancia focal de cada una de las cámaras.

- Calibración de la inclinación. Permite calcular la inclinación de cada una de las cámaras.

9.1.2.2. La aplicación

Para ejecutar la aplicación solo hace falta clickear en esta. Si no se tiene la aplicación en el ordenador, para instalarla tan solo se deberá ejecutar el instalador y seguir los pasos que se indican. Asegúrese de tener instalado el FrameWork .NET 2.0 o superior y el FrameWork Microsoft Visual C++ 2008 SP1.

Modo Localización

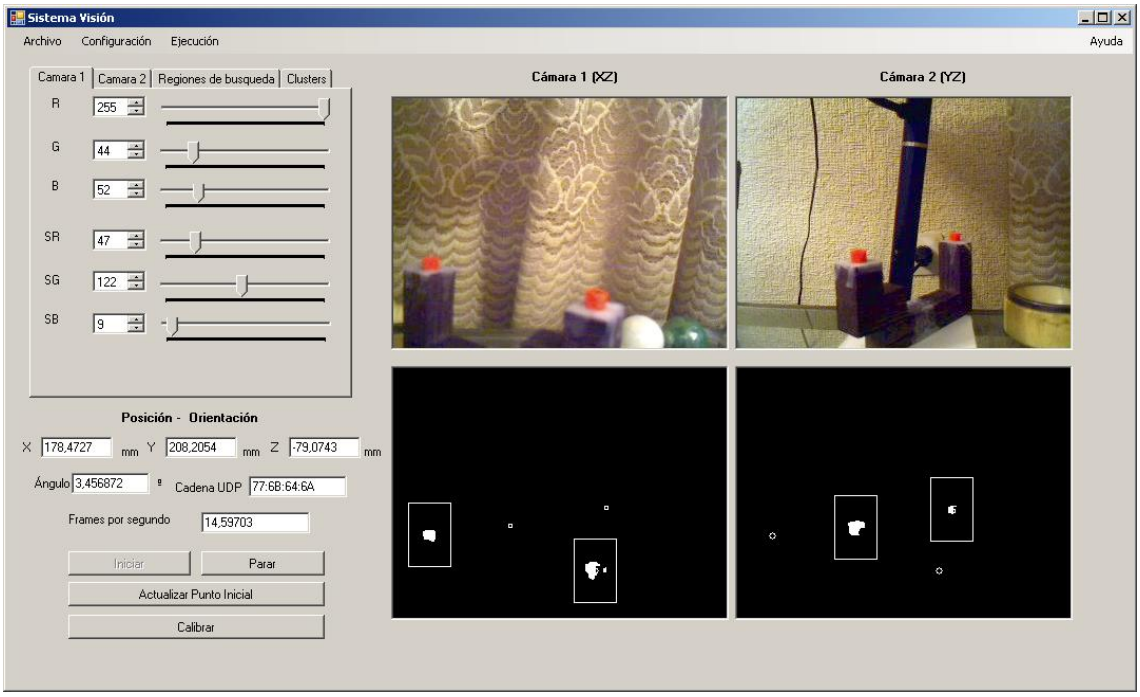


Figura 81: Interfaz gráfica con VS modo localización

Cuando se ejecute la aplicación (o después del modo calibración, que se verá a continuación) se entra en el modo localización. El programa en este modo muestra el aspecto de la Figura 81.

Como podemos comprobar en la parte derecha se ven las imágenes capturadas por las cámaras 1 y 2, debajo de estas imágenes se ven las capturas después de ser procesadas por los filtros.

En la parte izquierda hay unas pestañas en las que pone *Cámara 1* y *Cámara 2*. En dichas pestañas se encuentran unos trackbars permiten cambiar los colores de los filtros usados para cada cámara. Dependiendo del espacio de color en el que estemos trabajando tendrán un aspecto u otro, como se puede observar en la siguiente figura.

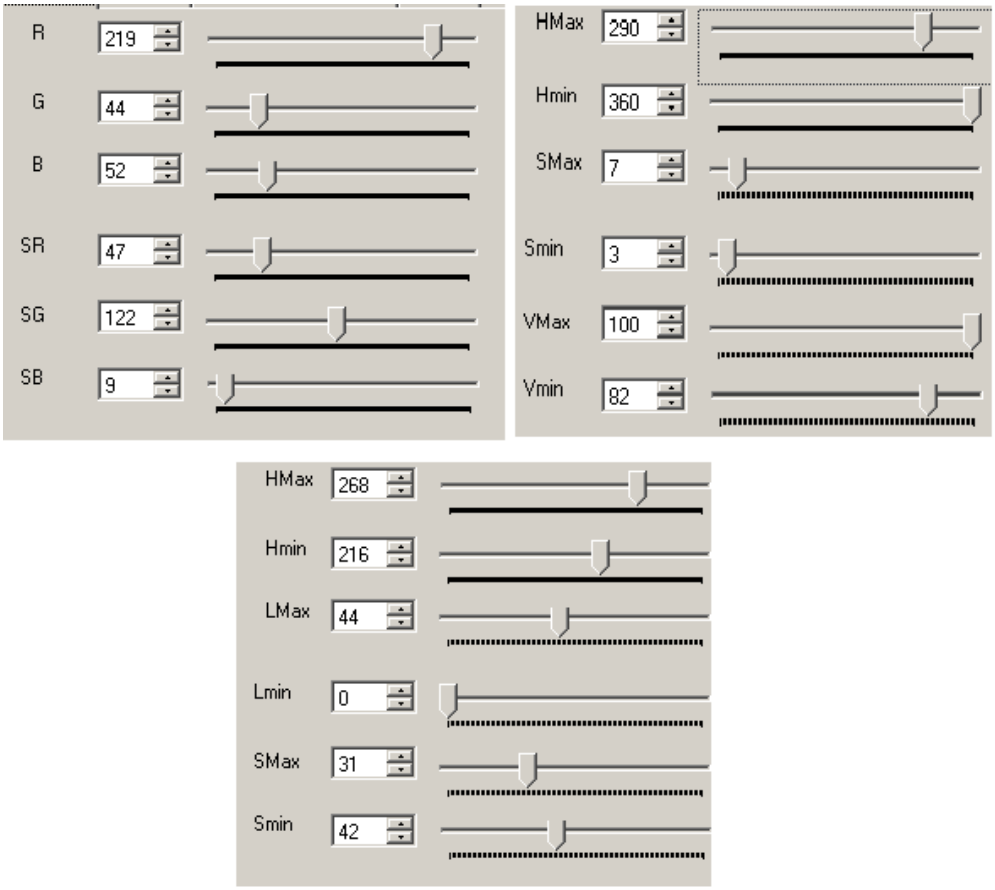


Figura 82: Distintos espacios de color

En la pestaña *Regiones de búsqueda* se permite indicar el ancho y alto de las regiones de búsqueda (véase Capítulo 4 apartado Localización) de cada una de las cámaras. El ancho y alto de cada región es independiente. En la siguiente figura se muestra la pestaña de modificación y como varían las regiones.

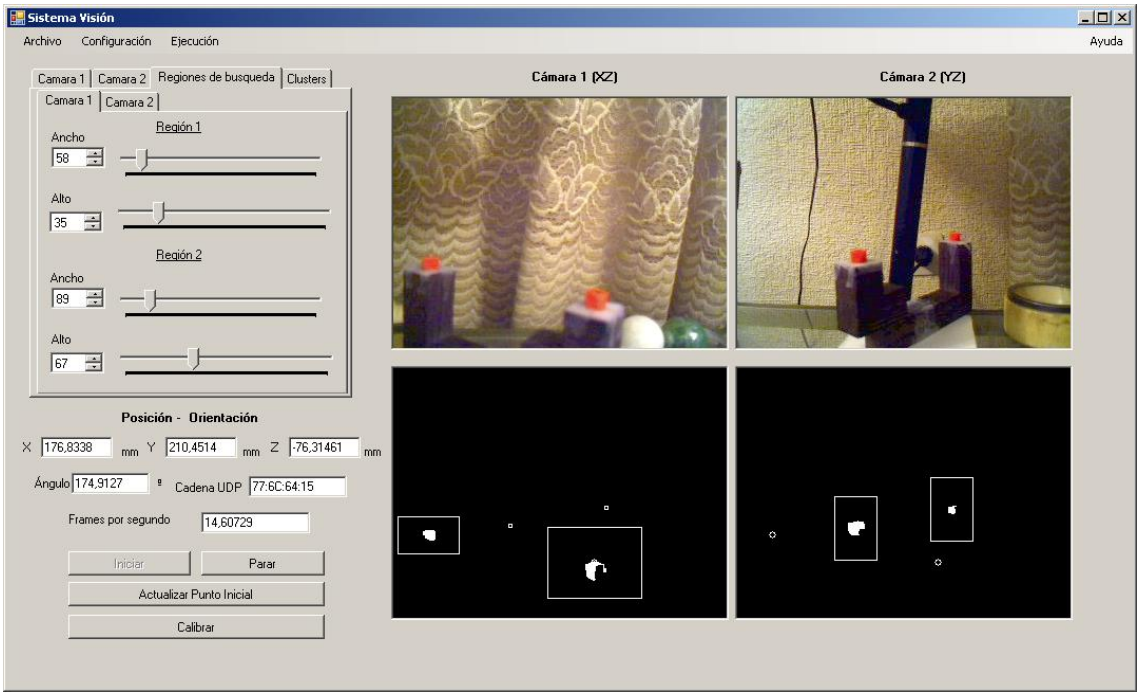


Figura 83: Modo localización pestaña de regiones de búsqueda

La última pestaña que queda por comentar es *clústeres*, donde que se puede modificar el área mínima de clúster y la distancia entre clúster de cada una de las cámaras (véase Capítulo 4 apartado Localización). La figura posterior nos muestra el aspecto de esta pestaña.

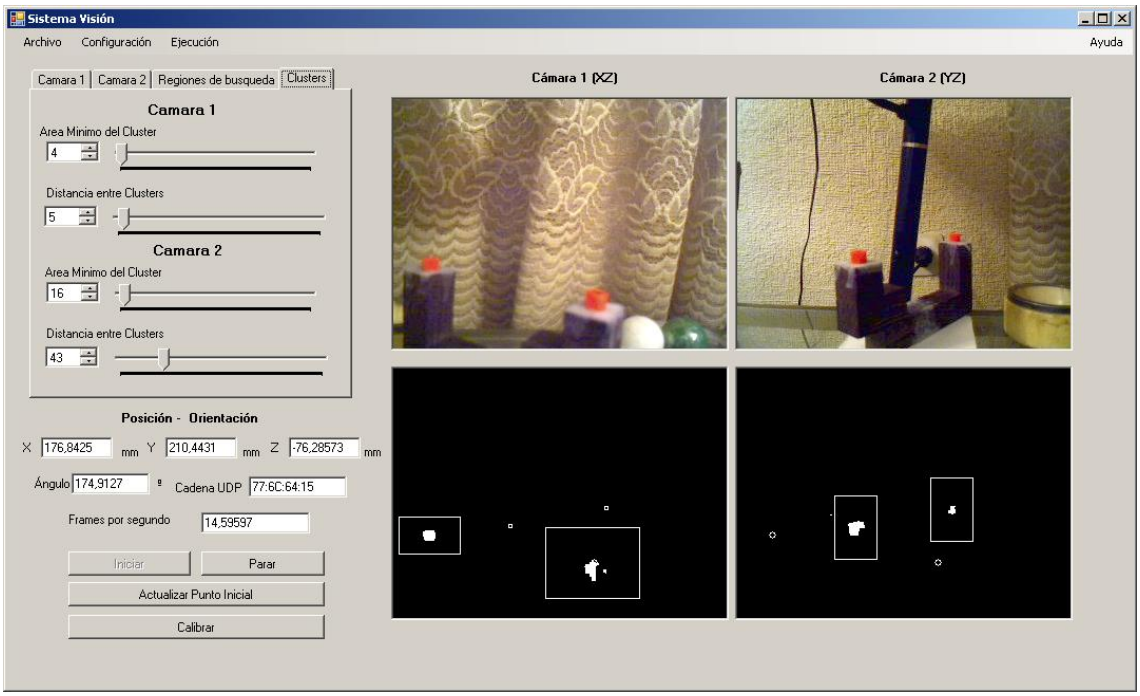


Figura 84: Modo localización pestaña de clústeres

Debajo de las pestañas anteriormente descritas se muestran los valores que se van obteniendo de la posición del objeto en el mundo real (X, Y, Z, ángulo). Si está activo el envío por UDP, se muestra la cadena que se está enviando, en hexadecimal, en un textbox. También se están mostrando los frames por segundo que se están siendo procesados.

Con el botón *Iniciar* se inicia la captura y procesamiento de imágenes, para terminar este proceso se usa el botón *Parar*. Mientras se está ejecutando el programa y esta iniciado se puede actualizar la posición inicial del objeto con el botón *Actualizar Punto Inicial*. Estas funcionalidades también se pueden activar con el menú *Ejecución* que nos ofrece las mismas características que los botones.

Con el botón *Calibrar* se pasaría al siguiente modo de ejecución.

Además de la funcionalidad básica se ha añadido alguna funcionalidad extra para que el usuario se sienta más cómodo al usar la aplicación y no tenga que estar cambiando a mano parámetros del XML.

Se permite cargar el XML con la opción *Cargar* del menú *Archivo*. Este menú también permite guardar el XML tanto con el nombre que ya tuviera (*Archivo -> Guardar)* como renombrándolo e incluso cambiándolo de ruta (*Archivo -> Guardar Como*).

Para cambiar algunos parámetros del XML está la opción *Preferencias* en el menú *Configuración* que permite cambiar datos como:

- Posición inicial de las cámaras.

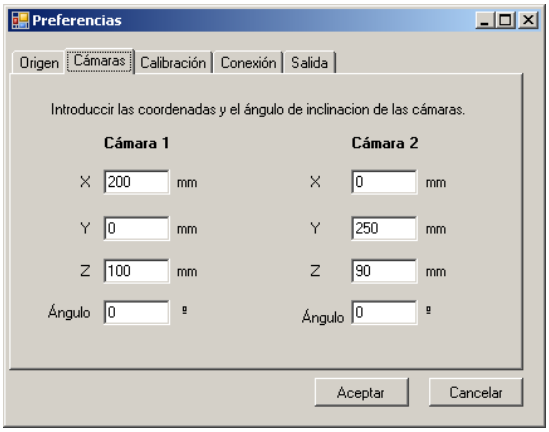


Figura 85: Ventana preferencias pestaña cámaras

- Nombre del fichero donde se guardan los datos, separador que se usan para los datos y el número de datos guardados.

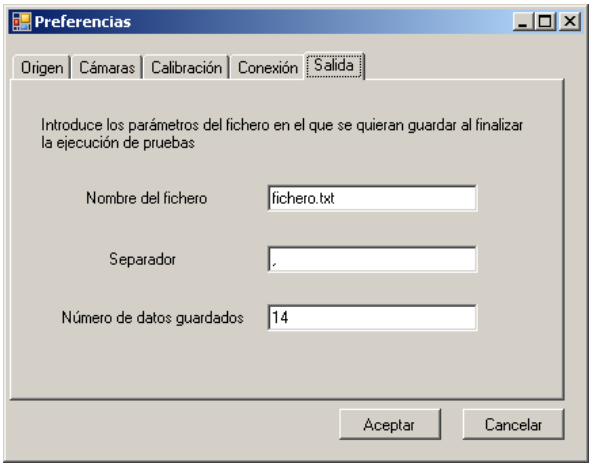


Figura 86: Ventana preferencias pestaña salida

- Distancia focal de las cámaras y el valor del pixel.

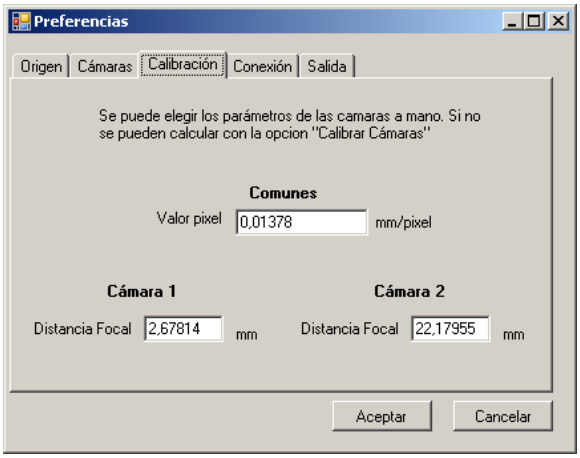


Figura 87: Ventana preferencias pestaña calibración

- Posición inicial del objeto. Si ésta es conocida la puedes introducir a mano.

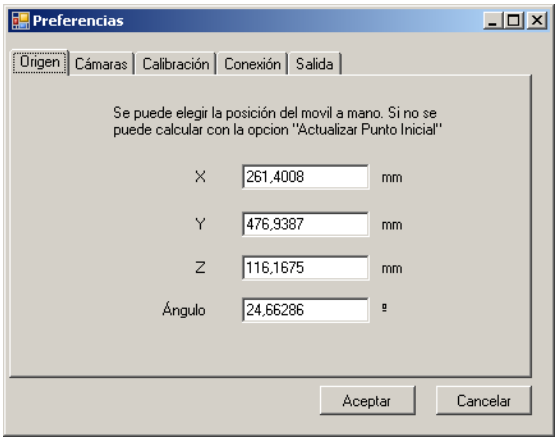


Figura 88: Ventana preferencias pestaña origen

- Parámetros de conexión, como la dirección IP del destino y el puerto usado.

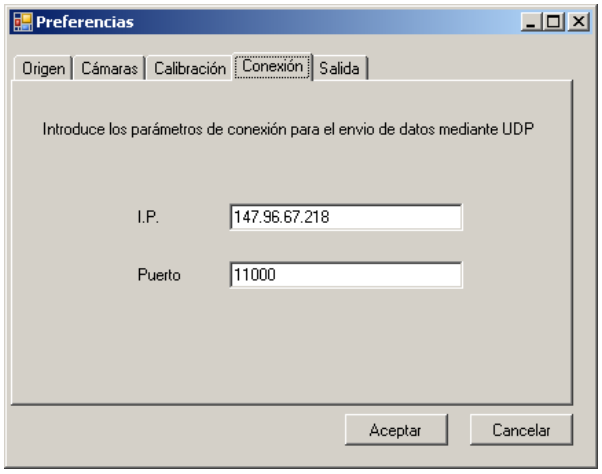


Figura 89: Ventana preferencias pestaña conexión

Las otras opciones de este menú son:

- *Guardar en fichero*: si no estaba activa la opción de guardar en fichero la activa y comienza a guardar los datos en fichero.
- *Enviar por UDP*: activa la opción de enviar por UDP.
- *Espacio de color*: permite al usuario elegir el espacio de color que se desee utilizar entre RGB, HSV y HLS.

También se ha añadido un menú ayuda con indicaciones sobre cada uno de los modos y de las pantallas en las que nos encontremos.

Modo Calibración

La primera pantalla de calibración da alguna indicación sobre la calibración y permite elegir qué modo de calibración se desea: calibración de la distancia focal o calibración de la inclinación (véase Figura 90).

En cualquier caso está la opción *Volver atrás* con lo que se pasara a la ventana anterior (ventana localización).

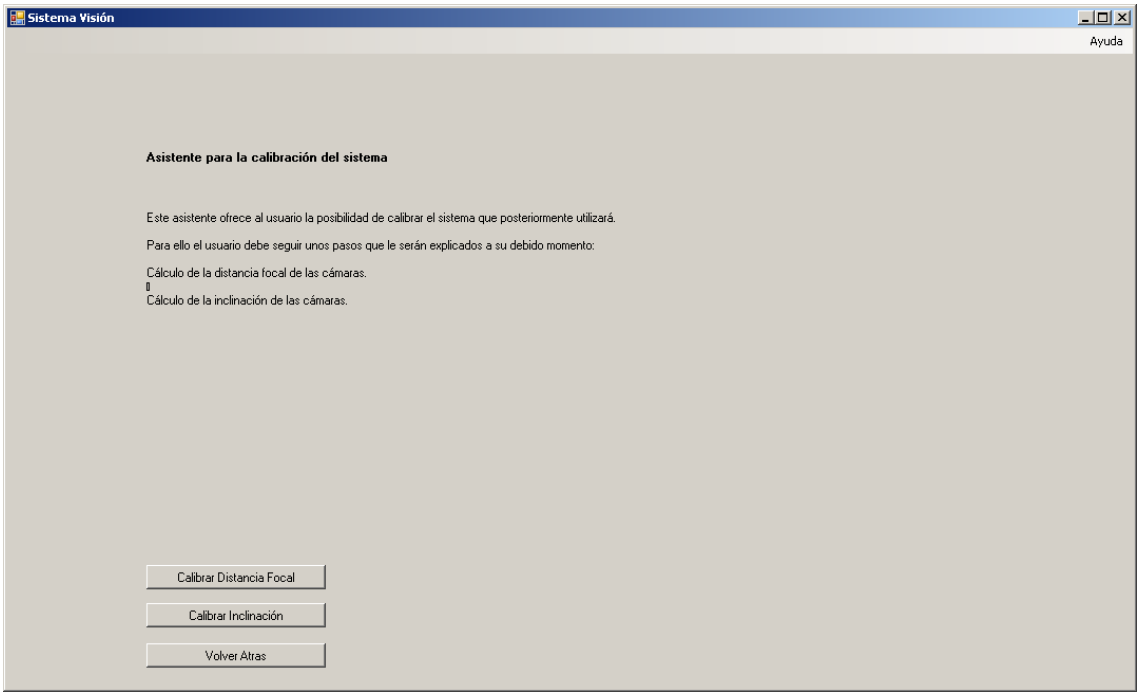


Figura 90: Introducción al modo calibración

Siendo indiferente que opción de calibración se haya escogido, la siguiente pantalla permite calibrar los colores de las cámaras para que solo se encuentren los clústeres deseados, así como ponerlas rectas usando algún patrón, evitando fallos en la calibración porque las cámaras estén ladeadas (véase Figura 91).

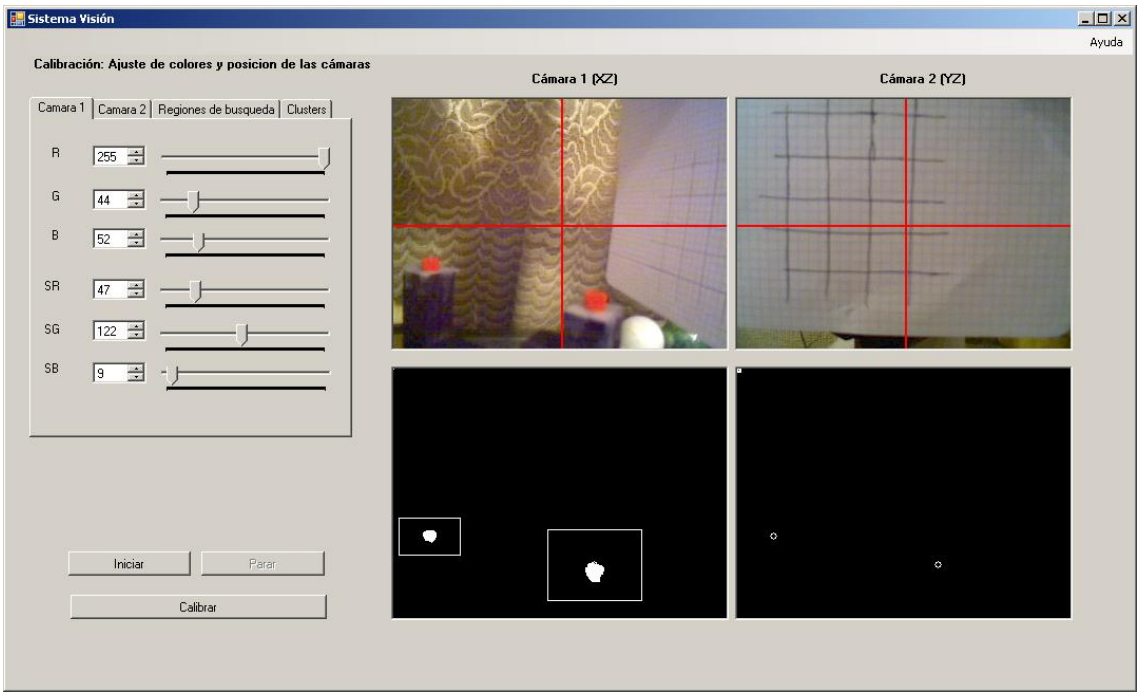


Figura 91: Modo calibración de colores

Calibración de la distancia focal

- Lo primero se rellenaran los textbox con las coordenadas de cada uno de los clúster.
- Después se dará a *Iniciar*, cuando se vea que los clústeres se ven bien en ambas cámaras se debe dar a *Parar*.
- Ahora se pulsa el botón *Calcular distancia focal*, que nos dará valores de las distancias focales de cada cámara, así como de error cometido. Este proceso se puede repetir hasta que los valores sean los deseados en cuyo caso daremos a *Guardar*. Esta opción guarda los valores de las distancias focales en el XML y son los que posteriormente utilizará. Después de darle a este botón vuelve al modo localización.
- En cualquier caso está la opción *Volver atrás* con lo que no se guardarían los datos y se pasaría a la ventana anterior (calibración inicial).

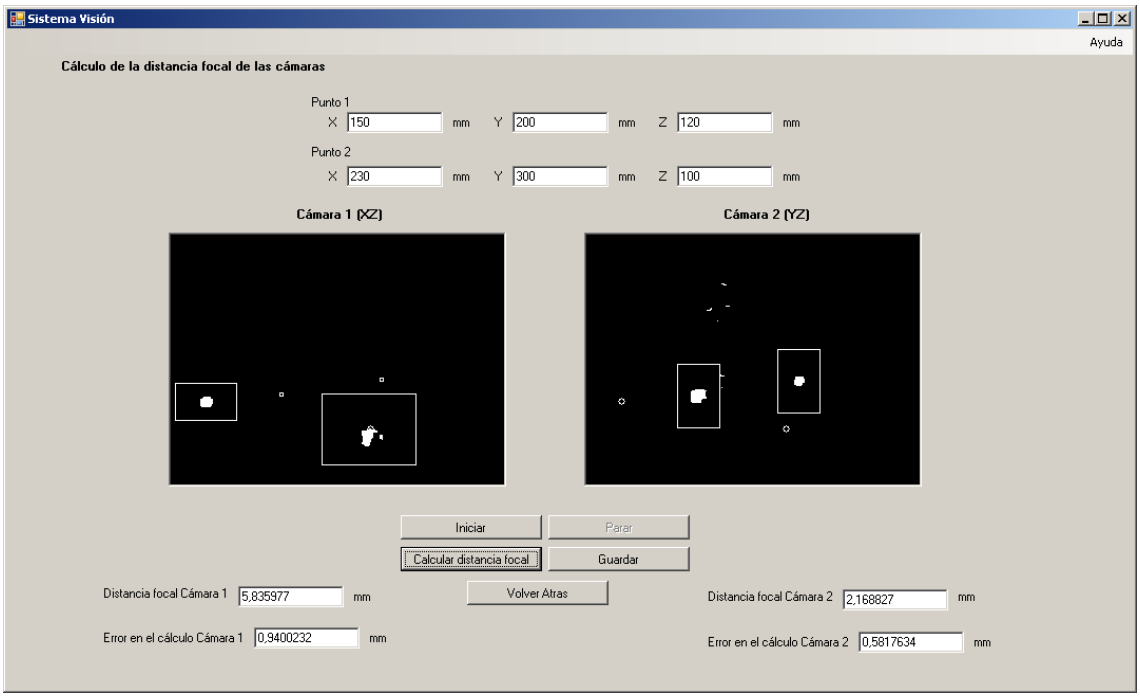


Figura 92: Modo calibración de la distancia focal

Calibración de la inclinación

- Lo primero se rellenaran los textbox con las coordenadas del clúster.
- Después se dará a *Iniciar*, cuando se vea que el clúster se ven bien y está centrado (donde el circulito blanco) se deberá *Parar*.
- Ahora se pulsa el botón *Calcular inclinación Cámara 1*, que nos dará el valor de la inclinación de la cámara 1.
- Se repetirá el proceso para la cámara 2.

- Este proceso se puede repetir hasta que los valores sean los deseados en cuyo caso daremos a *Guardar*. Esta opción guarda los valores de las inclinaciones en el XML y son los que posteriormente utilizará. Después de darle a este botón vuelve al modo localización.
- En cualquier caso está la opción Volver atrás con lo que no se guardarían los datos y se pasaría a la ventana anterior (calibración inicial).

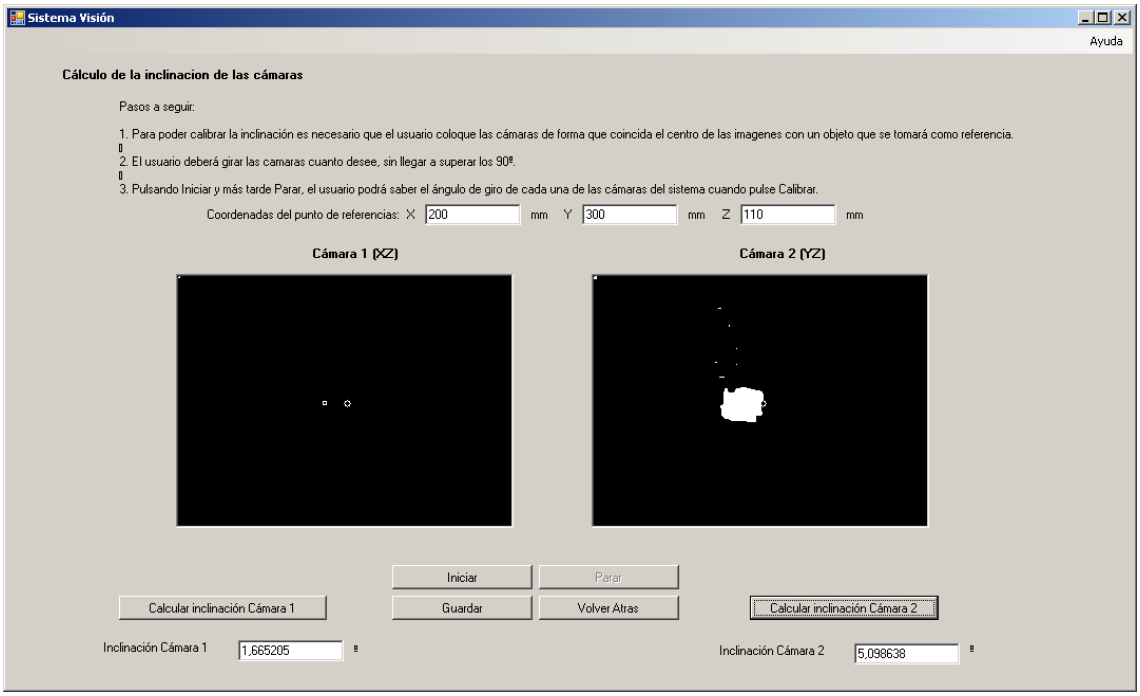


Figura 93: Modo calibración de la inclinación

9.1.3. Manual de usuario para el Receptor UDP

9.1.3.1. Introducción

El receptor UDP es un programa ajeno al sistema de visión. Se encarga de la recepción de datos enviados por el sistema de visión. Estos datos se muestran por pantalla pudiéndose dibujar para ver el recorrido del móvil que se esté estudiando. Tiene la posibilidad de almacenar los datos recibidos en un fichero para que posteriormente sean analizados por el usuario.

Su finalidad es doble, por una parte probar la transmisión por UDP y, por otra parte observar el funcionamiento del sistema de visión al conectarlo a una aplicación externa.

9.1.3.2. La aplicación

Para la ejecución de la aplicación solo hay que hacer doble clic en *Receptor.jar* o introducir por línea de comandos *java -jar Receptor.jar* Es importante que en el

mismo directorio de la aplicación este la capeta *lib* que contiene las librerías *core.jar y swing-layaut_1.0.3.jar*. Para mayor comodidad del usuario se ha incorporado un instalador con la aplicación que contiene las librerías anteriormente indicadas. Cuando iniciemos el programa nos saldrá una ventana como la mostrada en la imagen siguiente.

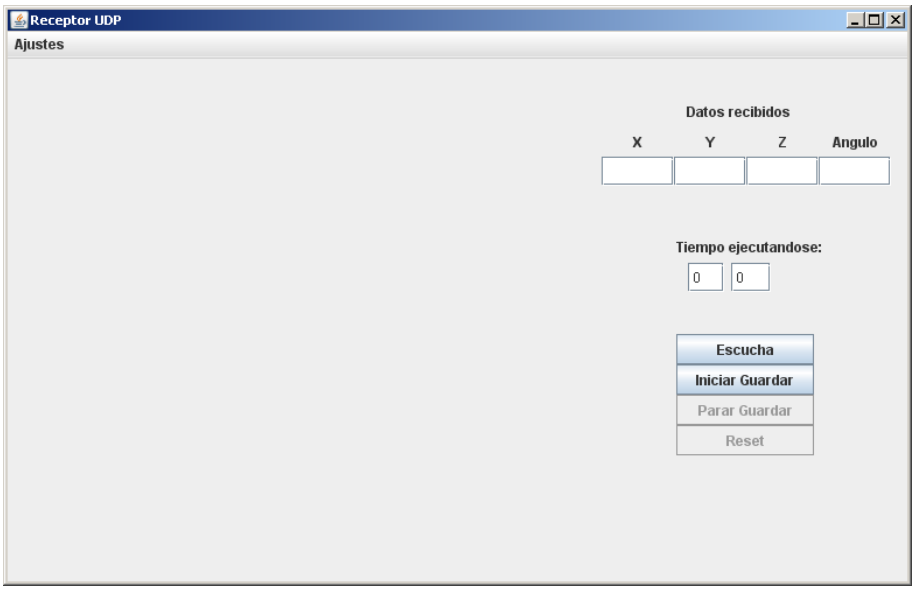


Figura 94: Receptor UDP

Antes de iniciar la ejecución se debería ajustar algunos parámetros (si no se ajustan se tomarían los valores por defecto que se indican a continuación). Tales parámetros son (véase Figura 95):

- Puerto: indica el puerto de recepción de los datos recibidos. Al usarse el protocolo UDP el número de puerto debe ser el mismo que en el servidor. Por defecto es el puerto 1234.
- Posición Inicial: se refiere a la posición inicial del móvil, tomándose esta como posición origen para el dibujado. Por defecto es la posición X=0, Y=0, Z=0 y Ángulo=0.
- Guardar en: permite guardar un archivo con los datos recibidos en la carpeta indicada con el nombre de archivo indicado. Por defecto se guarda en la misma carpeta que la aplicación con el nombre *recibido.txt*.

Se puede ajustar el numero de parámetros que se crea necesario así como no ajustar ninguno y dejar que la aplicación tomo los valores por defecto.

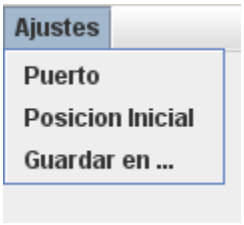


Figura 95: Menú ajustes del receptor UDP

Cuando se hayan ajustado los parámetros se puede iniciar la ejecución. Tenemos dos modos de ejecución, escuchar e iniciar guardar:

- Escuchar: permite ver los datos recibidos (X, Y, Z, ángulo) en los textbox. No dibuja el recorrido ni guarda los datos en el archivo. Aunque este en este modo se puede pasar al otro dándole en el botón *Iniciar Guardar* (véase Figura 96).

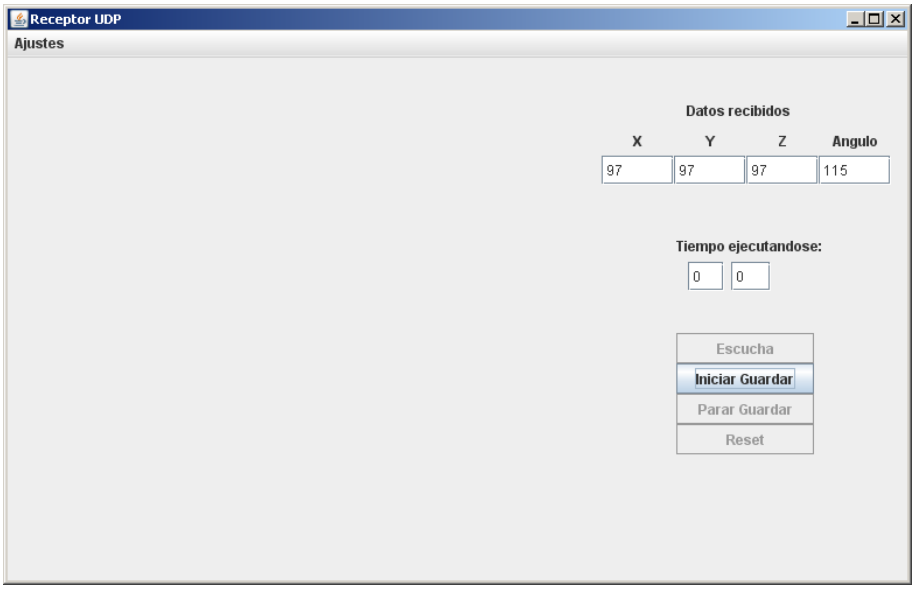


Figura 96: Modo escucha del receptor UDP

- Iniciar Guardar: Se muestran los datos recibidos como en el modo anterior, pero adicionalmente los guarda en un fichero junto al tiempo de recepción en milisegundos. Con este método se visualiza en un sistema de ejes el recorrido del objeto (véase Figura 97).

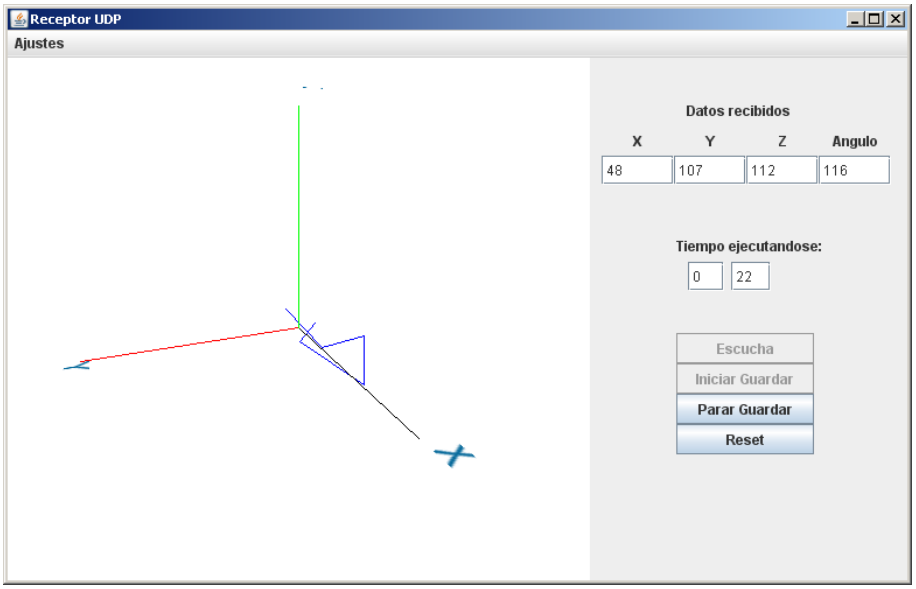


Figura 97: Modo iniciado del receptor UDP

Para parar la ejecución del modo guardar solo hay que darle al botón *Parar Guardar*, con lo que se deja de dibujar, de recibir y obviamente de escribir en el fichero. Si posteriormente se vuelve a iniciar guardar los datos recibidos se escriben al final del fichero sin borrarlo. También se puede volver a la escucha después de parar.

Las acciones que se llevan a cabo con el botón *Reset* son diferentes de las de parar. Además de lo que hace este ultimo botón (para la escucha, dejar de escribir en fichero y de mostrar por pantalla) se reinicia la ejecución del programa, pudiendo volver a ajustar los parámetros. Si posteriormente se vuelve a dar a iniciar y no se ha cambiado el fichero donde se quiere guardar los datos anteriores se borran al ser una nueva ejecución del programa.

Visualización gráfica

Para una mayor comodidad del usuario el diagrama gráfico del recorrido del móvil no es estático, es decir, el usuario puede cambiar el punto de visión para poder ver en recorrido desde otra perspectiva. Para rotar la cámara hacia arriba o abajo se usaran las flechas (flecha arriba y flecha abajo). Análogamente para rotar la cámara, alrededor de la escena, hacia la izquierda o la derecha (véase Figura 98). Si lo que se desea es acercar o alejar la cámara se usará la rueda del ratón.

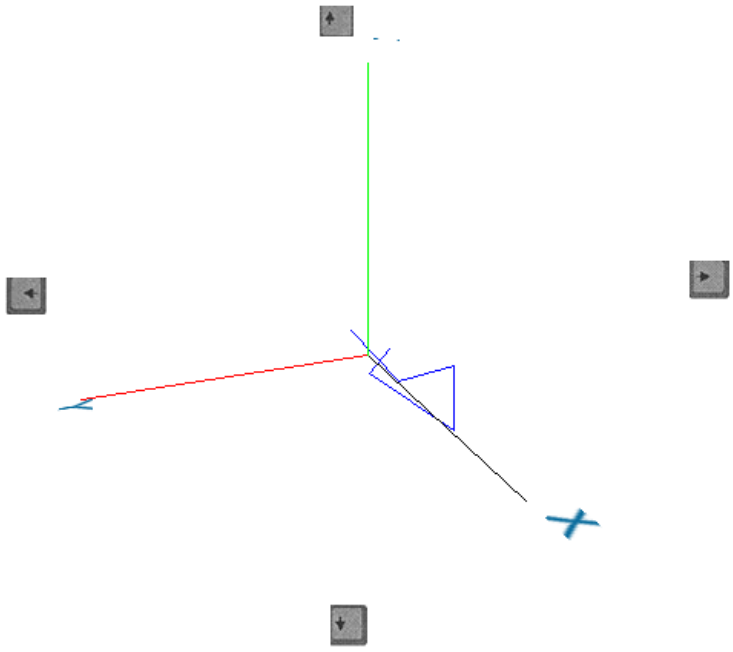


Figura 98: Controles de teclado para el movimiento de ejes

9.1.4. Documentación de la librería Motor

9.1.4.1. Descripción

La librería Motor contiene toda la funcionalidad necesaria para la localización de objetos en el espacio, haciendo uso de visión estereoscópica con dos cámaras posicionadas como está especificado en el Capítulo 2.

La librería se encarga tanto de la captura de las imágenes como su procesamiento y los cálculos necesarios para el sistema de localización. De manera que esto sea completamente transparente para el usuario.

La DLL motor también incluye la funcionalidad necesaria para la calibración de la distancia focal de las cámaras y la calibración de la inclinación.

9.1.4.2. Consideraciones de uso

Todas las funciones de la DLL Motor están compiladas de forma que usan el convenio de llamada STDCALL. Los usuarios de la dll deberán tenerlo en cuenta y utilizar las construcciones propias del lenguaje/entorno de desarrollo usado en caso de que éste utilice otra convención de llamada por defecto (CDECL en la mayoría de compiladores de C)

Junto a la DLL se incluye la librería estática (.LIB) que permite al compilador enlazar los programas a la DLL.

9.1.4.3. Referencia del API de Motor.dll

Referencia del API de Motor.dll (Por orden alfabético)

void calibraDistanciaFocal(float* val)	
Parámetros	
float* val	Puntero a un vector de 6 posiciones que contiene la posición de los dos elementos usados para la calibración de la distancia focal.
Descripción	
Esta función envía al motor los datos necesarios para la calibración de la distancia focal de las cámaras. El vector debe contener los siguientes valores *valores[0]* : Posición X del objeto de calibración 1 *valores[1]* : Posición Y del objeto de calibración 1 *valores[2]* : Posición Z del objeto de calibración 1 *valores[3]* : Posición X del objeto de calibración 2 *valores[4]* : Posición Y del objeto de calibración 2 *valores[5]* : Posición Z del objeto de calibración 2	

void calibraInclinacion(float* val)	
Parámetros	
float* val	Puntero a un vector de 3 posiciones que contiene la posición del elemento usados para la calibración de la inclinación de la cámara.
Descripción	
Esta función envía al motor los datos necesarios para la calibración de la inclinación de las cámaras. El vector debe contener los siguientes valores	

valores[0] : Posición X del objeto de calibración.

valores[1] : Posición Y del objeto de calibración.

valores[2] : Posición Z del objeto de calibración.

Para calibrar la inclinación de las 2 cámaras es necesario realizar dos llamadas a esta función pasando en cada llamada las posiciones obtenidas para la cámara correspondiente.

void cambiarCallback(int estado)	
Parámetros	
int estado	Nuevo modo de funcionamiento del motor.
Descripción	

Esta función cambia el modo de funcionamiento del motor dependiendo del valor del parámetro *estado*. Los valores posibles son los siguientes:

0 : Standby.

1 : Modo de localización / Calibrado de colores.

2 : Modo de calibración de distancia focal.

3 : Modo de calibración de inclinación.

void cambiaClusters(int* valores)	
Parámetros	
int* valores	Puntero a un vector de 12 posiciones que contiene los valores de configuración para el algoritmo de búsqueda de clústeres.
Descripción	

Esta función configura los parámetros del algoritmo de búsqueda de clústeres. Se espera que el vector de enteros tenga los siguientes valores:

valores[0] : Área mínima para clústeres detectados en la cámara 1

valores[1] : Distancia mínima entre clústeres detectados en la cámara 1

valores[2] : Alto de la región de búsqueda 1 para la cámara 1

valores[3] : Ancho de la región de búsqueda 1 para la cámara 1

valores[4] : Alto de la región de búsqueda 2 para la cámara 1

valores[5] : Ancho de la región de búsqueda 2 para la cámara 1

valores[6] : Área mínima para clústeres detectados en la cámara 2

valores[7] : Distancia mínima entre clústeres detectados en la cámara 2

valores[8] : Alto de la región de búsqueda 1 para la cámara 2

valores[9] : Ancho de la región de búsqueda 1 para la cámara 2

valores[10] : Alto de la región de búsqueda 2 para la cámara 2

valores[11] : Ancho de la región de búsqueda 2 para la cámara 2

Los cambios efectuados por esta función se aplican a la siguiente imagen capturada después de la ejecución de la misma.

void cambiaEspacio(int espacio)	
Parámetros	
int espacio	Identificador del espacio de color
Descripción	
Esta función cambia el espacio de color que usa el motor para procesar las	

imágenes. Los valores posibles de *espacio* son los siguientes:

0 : RGB.

1 : HSV

2 : HLS

void cambiaFiltros(int* valores)	
Parámetros	
int* valores	Puntero a un vector de 7 posiciones que contiene los valores de configuración para los filtros del sistema de localización referentes a la cámara 1
Descripción	

Esta función configura los parámetros de filtrado de color aplicables a las imágenes capturadas por la cámara 1. Se espera que el vector de enteros tenga los siguientes valores:

valores[0] : Modo de color (0=RGB,1=HSV,2=HLS)

El resto de componentes varían según el modo de color de la siguiente manera:

Modo RGB:

valores[1]: Componente R

valores[2]: Componente G

valores[3]: Componente B

valores[4] : Sensibilidad en R

valores[5] : Sensibilidad en G

valores[6] : Sensibilidad en B

Modo HSV:

valores[1]: Tono máximo

valores[2]: Tono mínimo

valores[3]: Saturación máxima

valores[4] : Saturación mínima

valores[5] : Valor máximo

valores[6] : Valor mínimo

Modo HLS:

valores[1]: Tono máximo

valores[2]: Tono mínimo

valores[3]: Luminosidad máxima

valores[4] : Luminosidad mínima

valores[5] : Saturación máxima

valores[6] : Saturación mínima

Los cambios efectuados por esta función se aplican a la siguiente imagen capturada después de la ejecución de la misma.

void cambiaFiltros2(int* valores)	
Parámetros	
int* valores	Puntero a un vector de 7 posiciones que contiene los valores de configuración para los filtros del sistema de localización referentes a la cámara 2
Descripción	

Esta función es análoga a cambiaFiltros pero se refiere a los parámetros de filtrado aplicados a la cámara 2. Para más información refiérase a la referencia de *cambiaFiltros*

void dameCadenaUDP(char* cad)	
Parámetros	
char* cad	Buffer de salida donde la función copiará el contenido del último paquete UDP con coordenadas que ha sido enviado por el motor de localización. En la implementación actual éste buffer de salida debe poder albergar al menos 5 char.
Descripción	
dameCadenaUDP recupera a un buffer local la última cadena de texto que ha sido enviada por el motor de localización por UDP al destinatario configurado en el fichero XML.	

void dameCalibracion(float* dFC1, float* sFC2, float* valorPixel)	
Parámetros	
float* dFC1	Buffer para la distancia focal de la cámara 1. Debe apuntar a memoria reservada.
float* dFC2	Buffer para la distancia focal de la cámara 2. Debe apuntar a memoria reservada.
float* valorPixel	Buffer para el valor del pixel. Debe apuntar a memoria reservada.
Descripción	
Obtiene del motor los datos referentes a la calibración las distancias focales de las cámaras (*dFC1 y dFC2)* y al valor del pixel (*valorPixel*).	

void dameClusters(int* valores)	
Parámetros	
int* valores	Puntero a un vector de 12 posiciones donde se copiarán los valores de configuración para el algoritmo de búsqueda de clústeres. Es importante que el vector esté reservado y en zona de memoria válida.
Descripción	
Esta función obtiene los parámetros del algoritmo de búsqueda de clústeres . Para más información sobre el contenido del vector refiérase a la documentación de la función *dameClusters*.	

void dameCoordenadas(float* coordenadas)	
Parámetros	
float* coordenadas	Puntero a un vector de 4 posiciones donde se devolverán las coordenadas donde ha sido detectado el objeto por el sistema de localización. El puntero tiene que apuntar a memoria reservada y por tanto a una posición válida.
Descripción	
Esta función obtiene las coordenadas calculadas por el sistema de localización para el objeto. El contenido del vector tras una llamada a la función es el siguiente: *coordenadas[0]* : Coordenada X del objeto *coordenadas[1]* : Coordenada Y del objeto *coordenadas[2]* : Coordenada Z del objeto *coordenadas[3]* : Ángulo de orientación del objeto	

void dameDistanciaFocal(float* val)	
Parámetros	
float* val	Puntero a un vector de 4 posiciones donde se copian los resultados de la calibración de distancia focal. El vector debe apuntar a una zona de memoria válida.
Descripción	
Esta función obtiene los resultados de una calibración de distancia focal. Después de llamarla *val* contiene: *val[0] :* Distancia focal de la cámara 1. *val[1] :* Error de calibración para la cámara 1. *val[2] :* Distancia focal de la cámara 2. *val[3] :* Error de calibración para la cámara 2.	

void dameFiltros(int* valores)	
Parámetros	
int* valores	Puntero a un vector de 7 posiciones donde se devolverán los valores de configuración para los filtros del sistema de localización referentes a la cámara 1. El puntero debe contener una dirección de memoria correcta.
Descripción	
Esta función obtiene los parámetros de filtrado de color aplicables a las imágenes capturadas por la cámara 1. El contenido del vector *valores* se corresponde a la definición contenida en la referencia de la función *cambiaFiltros.*	

void dameFiltros2(int* valores)	
Parámetros	
int* valores	Puntero a un vector de 7 posiciones donde se devolverán los valores de configuración para los filtros del sistema de localización referentes a la cámara 2. El puntero debe contener una dirección de memoria correcta.
Descripción	
Esta función obtiene los parámetros de filtrado de color aplicables a las imágenes capturadas por la cámara 2. Para más información ver documentación de *dameFiltros*.	

float dameFps()	
Valor devuelto	
float	Número de cuadros por segundo
Descripción	
Una llamada a esta función obtiene el número de imágenes por segundo que está capturando y analizando el sistema de localización.	

void dameInclinacion(float* val)	
Parámetros	
float* val	Puntero a un vector de 2 posiciones donde se copian los resultados de la calibración de inclinación de las cámaras. El vector debe apuntar a una zona de memoria válida.
Descripción	
Esta función obtiene los resultados de una calibración de inclinación. Después de llamarla *val* contiene:	

val[0] : Inclinación de la cámara 1. *val[1] :* Inclinación para la cámara 2.

void dameParametrosSalida(char* nombreFichero, char* separador, int* numDatos)	
Parámetros	
char* nombreFichero	Buffer para la ruta del fichero de salida. Debe apuntar a memoria reservada.
char* separador	Buffer para la cadena de separación de datos. Debe apuntar a memoria reservada.
int* numDatos	Buffer para el número de datos a guardar. Debe apuntar a memoria reservada.
Descripción	
Obtiene del motor las preferencias para el salvado de datos de localización en un TXT.	

void damePosicionCamaras(float* cam1, float* cam2)	
Parámetros	
float* cam1	Puntero a un vector de 4 posiciones que recibirá las componentes X,Y,Z, Ángulo de la cámara 1. El vector debe tener memoria reservada.
float* cam2	Puntero a un vector de 4 posiciones que recibirá las componentes X,Y,Z, Ángulo de la cámara 2. El vector debe tener memoria reservada.
Descripción	

Obtiene del motor las posiciones de las cámaras y las almacena en los vectores de salida *cam1* y *cam2*.

void damePosicionOrigen(float* val)	
Parámetros	
float* val	Puntero a un vector de 3 posiciones que recibirá las componentes X,Y,Z de la posición de origen. El vector debe tener memoria reservada.
Descripción	
Obtiene del motor las posiciones de origen y la almacena en el vector de salida *val*.	

void dameValConexion(char* ip, int* puerto)	
Parámetros	
char* ip	Buffer para la dirección IP o nombre de dominio donde se enviarán los paquetes de localización por UDP. Debe apuntar a memoria reservada con capacidad para albergar al menos 256 caracteres.
int* puerto	Buffer para el puerto al que se envían los paquetes UDP.
Descripción	
Obtiene el destinatario del envío de paquetes UDP con información sobre la localización del objeto.	

void enviarPorUDP(bool estado)	
Parámetros	
bool estado	*TRUE* si se desea que el programa envíe los resultados de localización por UDP, *False* en caso contrario.
Descripción	
Indica al motor si debe enviar cadenas UDP con los datos de localización.	

void guarda(char* rutaDif)	
Parámetros	
char* rutaDif	Cadena de texto con la ruta del archivo XML donde se guardarán las configuraciones. En el caso de que sea *NULL* se intentará guardar en la ruta de donde fueron cargadas por última vez.
Descripción	
Guarda las configuraciones del motor en un fichero XML indicado por el parámetro *rutaDif* o en el mismo fichero de donde se leyeron si no se especifica una nueva ruta.	

void guardarEnFichero(bool estado)	
Parámetros	
bool estado	*TRUE* si se desea que el programa guarde en fichero los resultados del modulo de localización , *FALSE en otro caso*
Descripción	
Indica al motor si debe guardar en un fichero TXT los resultados que va generando el sistema de localización.	

void iniciarCaptura(**pFuncionImagen** funcionImagen)	
Parámetros	
pFuncionImagen funcionImagen	Puntero a la función que recibirá las imágenes obtenidas y/o procesadas por el motor. La signatura de esta función debe ser la siguiente: ***void*** *fImagen(**IplImage*** *img,**int** id)* Donde: • img : Imagen generada por el motor • id : Identificador único del tipo de imagen El usuario de la dll debe implementar una función con dicha signatura ya que el parámetro no puede ser NULL
Descripción	
Esta función inicia el proceso de adquisición de imágenes y de su tratamiento. Tras iniciar la captura desde las dos primeras cámaras del sistema, coloca al sistema en modo Localización y comienza a procesar las imágenes adquiridas. Cuando el sistema tenga una nueva imagen disponible, ésta le será pasada a la función callback definida por el usuario en el parámetro *funcionImagen*.	

bool mostrarVentanas()	
Valor devuelto	
bool	*TRUE* si el motor está tiene activada la opción de mostrar ventanas usando Highgui, *FALSE* en caso contrario.

Descripción
Obtiene un booleano que indica si la DLL intentará mostrar ventanas o no.

void ponCalibracion(float dFC1, float sFC2, float valorPixel)	
Parámetros	
float dFC1	Distancia focal de la cámara 1
float dFC2	Distancia focal de la cámara 2
float valorPixel	Valor del pixel
Descripción	
Configura el motor para que considere como opciones de calibración las distancias focales de las cámaras (*dFC1 y dFC2)* y el valor del pixel (*valorPixel*).	

void ponInclinacion(float* val)	
Parámetros	
float* val	Puntero a un vector de 2 posiciones que contiene las inclinaciones de las dos cámaras.
Descripción	
Configura el motor para que considere que las inclinaciones de las 2 cámaras son las indicadas en las posiciones 0 y 1 de *val* respectivamente	

void ponParametrosSalida(char* nombreFichero, char* separador, int numDatos)
Parámetros

char* nombreFichero	Ruta del fichero de salida
char* separador	Cadena de separación de datos
int numDatos	Número de datos a guardar
Descripción	
Configura la manera en que el motor guarda en un TXT los datos de localización cuando la opción de guardar está activa. Se puede configurar la ruta del archivo, el separador entre datos y la longitud máxima del archivo.	

	void ponPosicionCamaras(float* cam1, float* cam2)
Parámetros	
float* cam1	Puntero a un vector de 4 posiciones con las componentes X,Y,Z, Ángulo de la cámara 1
float* cam2	Puntero a un vector de 4 posiciones con las componentes X,Y,Z, Ángulo de la cámara 2
Descripción	
Configura el motor para que considere que las posiciones de las cámaras son las indicadas por los vectores de entrada *cam1* y *cam2*.	

	void ponPosicionOrigen(float* val)
Parámetros	
float* val	Puntero a un vector de 3 posiciones que contiene las coordenadas de la posición origen.
Descripción	
Configura el motor para que considere que las coordenadas de la posición origen son las indicadas en el vector *val*.	

void ponPuntIni()
Descripción
ponPuntIni instruye al sistema de localización para que, en adelante, considere como punto inicial del sistema de referencia al punto actual donde se ha detectado el objeto.

bool ponRutaXML(char* cad)	
Parámetros	
char* cad	Cadena de texto con la ruta del XML que se desea cargar.
Valor devuelto	
bool	*TRUE* si se ha podido cargar el archivo. *FALSE* en caso contrario.
Descripción	
Esta función provoca que el motor recargue su configuración desde el fichero XML indicado por el parámetro *cad*.	

void ponValConexion(char* ip, int puerto)	
Parámetros	
char* ip	Dirección IP o nombre de dominio donde se enviarán los paquetes de localización por UDP.
int puerto	Puerto al que se envían los paquetes UDP.
Descripción	
Configura el destinatario del envío de paquetes UDP con información sobre	

la localización del objeto.

void terminarCaptura(**bool** guardar)	
Parámetros	
bool guardar	*True* si se desea guardar la configuración actual en el fichero XML correspondientes. *False* en otro caso
Descripción	
terminarCaptura provoca la parada completa del motor, libera los recursos reservados por la librería y fuerza un guardado de la configuración del sistema de localización en caso de que el parámetro *guardar* sea cierto.	

9.2. Apéndice B. Módulos internos de la librería Motor

9.2.1. Introducción

En este apartado se describen los módulos y funciones internas que se encuentran implementados en la librería Motor. Estos módulos y funciones son los que se utilizan cuando se realizan las llamadas a las funciones de la librería que se detallan en el Apartado A (Documentación de la librería Motor).

9.2.2. Descripción de los módulos

Módulo conexion

Funciones que se encargan del envío de los datos mediante un datagrama UDP. Se encargan de abrir y cerrar la conexión, y de crear el socket con la información obtenida del sistema de medida y de enviarlo a la IP y el puerto que se indiquen.

La funciones implementadas en este módulo son las siguientes:

- **comenzarConexion**
  - Descripción: Se encarga de abrir la conexión para poder comenzar a enviar los datagramas.
- **terminarConexion**
  - Descripción: Se encarga de cerrar la conexión antes de finalizar la ejecución de la aplicación.
- **enviarPuntoUDP**
  - Descripción: Se encarga de enviar la información calculada por el sistema a un computador de la red.
  - Parámetros de entrada: Cadena con las coordenadas y la orientación del objeto, e ip y puerto del computador al que se enviará la información.

Módulo config

Clases que se encargan del tratamiento del archivo xml de configuración. Al comienzo de la ejecución del sistema se crea un objeto de la clase Configuracion que será el que guarde los datos del xml durante la ejecución.

La clase principal de este módulo es la clase Configuración, que se encarga de guardar toda la información del xml en memoria. Los métodos de esta clase son los siguientes:

- **cargar**

- Descripción: carga el xml indicado en un objeto de la clase Configuración, se encarga de comprobar que todos los valores introducidos en el xml tienen el formato correcto y que se encuentran dentro de los intervalos aceptados.
- Parámetros de entrada: ruta o nombre del xml a cargar.
- Salida: booleano que indica si la carga se ha realizado correctamente.

– **guardar**
- Descripción: guarda un objeto de la clase Configuración en el archivo indicado.
- Parámetros de entrada: ruta o nombre donde se guardará el xml.
- Salida: booleano que indica si el guardado se ha realizado correctamente.

– **crear**
- Descripción: creo un objeto de la clase Configuración con los valores por defecto.
- Salida: booleano que indica si la creación se ha realizado correctamente.

Módulo salida

Función que se encarga del guardado de los datos obtenidos durante la ejecución en un archivo de texto al finalizar la aplicación.

La función implementada en este módulo es la siguiente:

– **guardarDatos**
- Descripción: Se encarga de guardar la información calculada durante la ejecución en un archivo.
- Parámetros de entrada: Nombre del archivo de salida, array con los datos a guardar, número de medidas totales, número de datos por medida y separador entre los datos de una misma medida.

Módulo tracking

En este módulo se encuentran las clases y funciones que se encargan de todos los procesos de cálculo relativos a la localización y a la calibración de las cámaras. Además, también se encuentran funciones encargadas de binarizar las imágenes y de la búsqueda de los objetos de referencia en ellas.

Matriz

Clase que implementa los cálculos necesarios sobre matrices, destacando la multiplicación, inversa y determinante.

Punto

Clase que implementa los cálculos necesarios sobre puntos, como el cálculo de la distancia entre puntos y el paso del punto de coordenadas locales a globales.

Vector

Clase que implementa los métodos necesarios sobre vectores, como ver si dos vectores son proporcionales.

Recta

Clase que implementa los métodos necesarios sobre rectas, como la obtención del punto de corte aproximado entre dos rectas y su distancia mínima.

TrackingUtils

Funciones que se encargan principalmente de la obtención del punto de la localización y de la obtener los valores de la calibración de las cámaras.

Las funciones implementadas en este módulo son:

- **obtenerPunto**
  - Descripción: Obtiene las coordenadas absolutas en las que se encuentra el objeto de referencia.
  - Parámetros de entrada: Coordenadas del objeto de referencia en las imágenes, matrices de transformación y parámetros de las cámaras.
  - Salida: Punto del objeto de referencia.
- **matrizTransformacion**
  - Descripción: Se encarga de obtener la matriz de transformación de una cámara.
  - Parámetros de entrada: Número identificador de la cámara y parámetros utilizados para crear la matriz de transformación.
  - Salida: Matriz de transformación de la cámara.
- **calibracionDistFocal**
  - Descripción: Obtiene la distancia focal de una cámara.
  - Parámetros de entrada: Puntos con las coordenadas de los objetos de referencia y coordenadas en las imágenes, número identificador de la cámara y sus parámetros.
  - Salida: Valor de la distancia focal y error obtenido en el cálculo.
- **calibracionInclinacion**
  - Descripción: Obtiene la inclinación de una cámara.
  - Parámetros de entrada: Punto con las coordenadas del objeto de referencia y coordenadas en la captura, y ancho y alto de la imagen.
  - Salida: Ángulo de inclinación de la cámara.
- **calcularPosicionRelativa**
  - Descripción: Calcula la posición relativa respecto a una posición de origen.
  - Parámetros de entrada: Coordenadas y ángulo de origen y los absolutos obtenidos de la localización.
  - Salida: Coordenadas y ángulo relativos.

CommonUtils

Funciones que se encargan del filtrado las imágenes para realizar su binarización y de la búsqueda de los objetos mediante el uso de clústeres.

Las funciones implementadas en este módulo son:

- **CvFindColorRGB**
  - Descripción: Se encarga de filtrar la imagen en el espacio de color RGB para devolver una imagen binarizada que tendrá en blanco los objetos del color introducido como parámetros y en negro el resto.
  - Parámetros de entrada: Imágenes fuente y destino y parámetros que indican el color y la sensibilidad deseadas.
- **CvFindColorHSV**
  - Descripción: Se encarga de filtrar la imagen en el espacio de color HSV para devolver una imagen binarizada que tendrá en blanco los objetos del color introducido como parámetros y en negro el resto.
  - Parámetros de entrada: Imágenes fuente y destino y parámetros que indican los umbrales mínimo y máximo del color deseado.
- **CvFindColorHLS**
  - Descripción: Se encarga de filtrar la imagen en el espacio de color HLS para devolver una imagen binarizada que tendrá en blanco los objetos del color introducido como parámetros y en negro el resto.
  - Parámetros de entrada: Imágenes fuente y destino y parámetros que indican los umbrales mínimo y máximo del color deseado.
- **buscaCluster**
  - Descripción: Función que implementa el algoritmo de búsqueda de clústeres (véase Capítulo 4 Apartado Localización) en una región concreta, pudiendo ser ésta la imagen completa.
  - Parámetros de entrada: Configuración del sistema, imagen filtrada, identificador de la cámara que ha dado la imagen, región en la que buscar, modo de búsqueda.
  - Parámetros de entrada/salida: Número de clústeres y lista de clústeres.
  - Salida: True si se ha encontrado un único clúster en la región de búsqueda, false en caso contrario.
- **buscaCoord**
  - Descripción: Función que decide en que región buscar y cuando. Si está buscando en dos regiones y no se encuentra el número de clústeres necesarios ejecuta la búsqueda global. Se encarga además de mantener las regiones de búsqueda.
  - Parámetros de entrada: Configuración del sistema, regiones de búsqueda e imágenes filtradas.
  - Salida: Puntos en los que se encuentran los objetos de referencia en las imágenes.
- **obtenerCoord**

- Descripción: Función que realiza las llamadas pertinentes a buscaCoord para realizar el proceso para ambas cámaras.
- Parámetros de entrada: Configuración del sistema e imágenes filtradas.
- Salida: Puntos en los que se encuentran los objetos de referencia en las imágenes.

– **procesarImagenes**
- Descripción: Función que se encarga de realizar el procesado de las imágenes capturadas por las cámaras.
- Parámetros de entrada: Configuración del sistema y capturas obtenidas de las cámaras.
- Salida: Imágenes volteadas, imágenes sin ruido, imágenes binarizadas e imágenes finales tras el procesado.

Módulos encargados de las distintas funcionalidades

CalcPosInicial

Funciones que se encargan del tratamiento de las imágenes para calcular la posición inicial del objeto que será guardada en el xml de configuración para su posterior utilización en la localización.

CalibDistFocal

Funciones que se encargan del tratamiento de las imágenes para realizar los cálculos necesarios para hallar la distancia focal de cada una de las cámaras.

CalibInclinacion

Funciones que se encargan del tratamiento de las imágenes para realizar los cálculos necesarios para hallar la inclinación de cada una de las cámaras.

Localizacion

Funciones que se encargan del tratamiento de las imágenes para realizar el proceso de localización del objeto.

Motor

Funciones que se encargan de la captura de imágenes, de su tratamiento y de las llamadas pertinentes al modulo que este activo. Este módulo también se encarga de la comunicación con el GUI.

9.3. Apéndice C. Herramientas de desarrollo

9.3.1. Descripción de las herramientas

9.3.1.1. OpenCV

OpenCV (Open source Computer Vision Library) es una librería de código abierto de visión artificial originalmente desarrollada por Intel. Está escrita en C y C++ y es multiplataforma, pudiendo ser usada en Windows, Linux y Mac OS X.

Está orientada principalmente al tratamiento de imágenes en tiempo real, implementando una gran variedad de herramientas diseñadas para ser computacionalmente eficientes para el tratamiento de imágenes. Si se deseara una mayor optimización sobre arquitecturas Intel, proporciona una interfaz a la librería Intel's Integrated Performance Primitives (IPP), que consiste en rutinas de bajo nivel optimizadas en muchas diferentes áreas algorítmicas.

Una de las metas de OpenCV es proveer una infraestructura de fácil uso para ayudar a crear aplicaciones de visión sofisticadas rápidamente, proporcionando unas 500 funciones que pueden ser aplicadas en muchas áreas de visión artificial, incluyendo inspección de producción en fábricas, tratamiento de imágenes médicas, seguridad, calibración de cámaras, visión estéreo y robótica.

Página oficial: http://sourceforge.net/projects/opencvlibrary/

9.3.1.2. MinGW

MinGW (Minimalist GNU for Windows) es una versión nativa del compilador GNU Compiler Collection (GCC) para la plataforma Windows, además de un conjunto de bibliotecas, archivos de encabezado y herramientas de desarrollo, lo cual proporciona un ambiente completo que permite a los desarrolladores crear aplicaciones nativas en Windows.

Página oficial: http://www.mingw.org/

9.3.1.3. Eclipse

Eclipse es un entorno de desarrollo integrado (IDE) de código abierto multiplataforma. Está escrito principalmente en Java y es usado para desarrollar aplicaciones en este lenguaje, pero mediante la instalación de plugins, puede ser utilizado para desarrollar aplicaciones en otros lenguajes como C, C++, Cobol, Pitón, Perl, PHP, etc.

Página oficial: http://www.eclipse.org/

9.3.1.4. CDT

El CDT (C/C++ Development Tools) Project proporciona una funcionalidad completa de un entorno de desarrollo integrado de C y C++ para la plataforma Eclipse.

Página oficial: http://www.eclipse.org/cdt/

9.3.1.5. VideoInput

Es una librería de código abierto que facilita la captura de imágenes disponible para Windows. Soporta multithreading y captura de múltiples dispositivos simultáneamente. Las imágenes capturadas mediante esta librería son compatibles con OpenCV.

Página oficial: http://muonics.net/school/spring05/videoInput/

9.3.1.6. Visual Studio 2008

Es un entorno de desarrollo diseñado por Microsoft para aplicaciones Windows, permite un desarrollo de aplicaciones rápido en varios lenguajes de programación tales como C++, C#, Visual Basic. Visual Studio 2008 permite crear de una manera cómoda aplicaciones visuales. Ofrece también funciones para bases de datos y el soporte necesario para crear aplicaciones AJAX.

Página oficial: http://www.microsoft.com/spanish/msdn/latam/visualstudio2008/

9.3.1.7. Netbeans 6.1

Netbeans es un entorno de desarrollo para Java, que permite hacer fácilmente interfaces gráficas. Es multiplataforma, gratuito y de código abierto. Actualmente soporta otros lenguajes como C, C++, PHP, Java Script...

Página oficial: http://www.netbeans.org/

9.3.1.8. HTML Help WorkShop

Es una herramienta gratuita de Microsoft que permite generar ficheros de ayuda en formato HTML compilado a partir de ficheros HML y CSS estándar. Con esta herramienta se ha diseñado en interfaz en .NET.

Página oficial:

http://www.microsoft.com/downloads/details.aspx?FamilyID=00535334-c8a6-452f-9aa0-d597d16580cc&displaylang=en

9.3.1.9. Processing V1.0

Es una librería gratuita de código abierto para Java que facilita el dibujado de modelos 3D y su animación. Se ha usado en el receptor UDP para mostrar la trayectoria del objeto.

Página oficial: http://processing.org/

9.3.2. Instalación de las herramientas

9.3.2.1. OpenCV

Descargar el instalador de la versión 1.0 para Windows de la sección de descargas de la página oficial del proyecto:

http://sourceforge.net/project/showfiles.php?group_id=22870&package_id=16937

Instalar las librerías siguiendo los pasos marcados.

Actualizar el path en las variables de entorno (Panel de control -> Sistema -> Opciones avanzadas -> Variables de entorno -> Variables de sistema -> Path) añadiendo:

;C:**<directorio de OpenCV>**\bin

9.3.2.2. MinGW

Descargar el instalador automático de la última versión de MinGW de la sección de descargas de la página:

http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=240780

Ejecutar el instalador.

Seleccionar Download and install.

Figura 99: Instalación MinGW 1

Seleccionar Current.

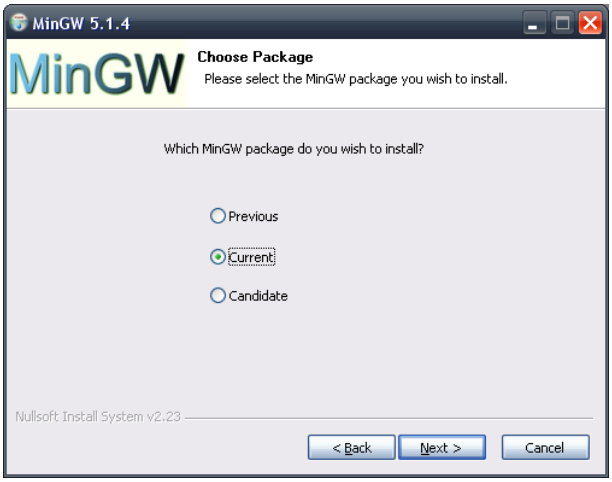


Figura 100: Instalación MinGW 2

Seleccionar g++ compiler.

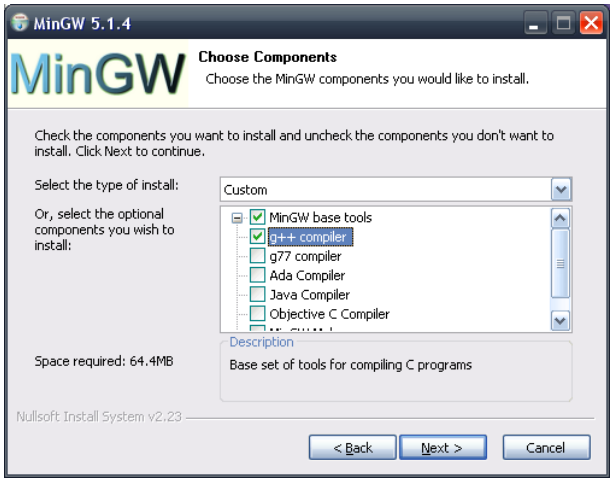


Figura 101: Instalación MinGW 3

Finalizar la instalación.

Actualizar el path en las variables de entorno añadiendo:

;C:\MinGW\bin

Descargar la última versión del depurador gdb (GNU debugger) de la sección de descargas de la página del MinGW:

http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=20507

Descomprimir el archivo en la carpeta donde se ha instalado el MinGW.

Descargar la última versión del GNU make de la sección de descargas de la página del MinGW:

http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=23918

Descomprimir el archivo en la carpeta donde se ha instalado el MinGW.

9.3.2.3. Eclipse

Descargar la última versión para Windows de Eclipse Classic que se puede encontrar en la página de descargas de Eclipse:

http://www.eclipse.org/downloads/

Descomprimir el programa en la carpeta que se desee.

9.3.2.4. CDT

Abrir Eclipse y seleccionar Help -> Software Updates... -> Available Software -> Add Site...

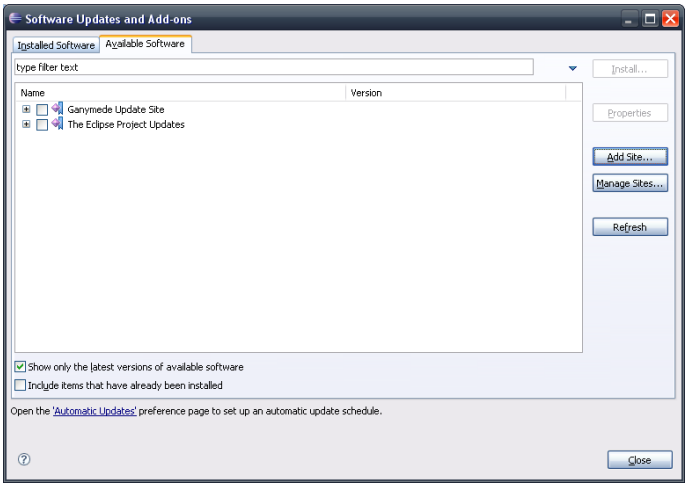


Figura 102: Instalación CDT 1

Introducir la dirección:

http://download.eclipse.org/tools/cdt/releases/ganymede

Esta dirección es la versión de CDT para la versión 3.4 de Eclipse, para futuras versiones de Eclipse ir a la página de CDT (http://www.eclipse.org/cdt/downloads.php) para ver cuál es la versión necesaria.

Activar el checkbox de la dirección introducida, dar a Install… y en la ventana emergente seguir los pasos dar a Finish para confirmar la instalación.

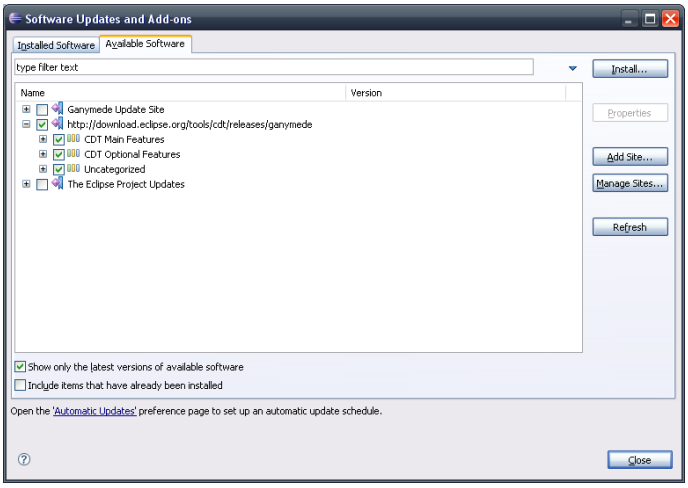


Figura 103: Instalación CDT 2

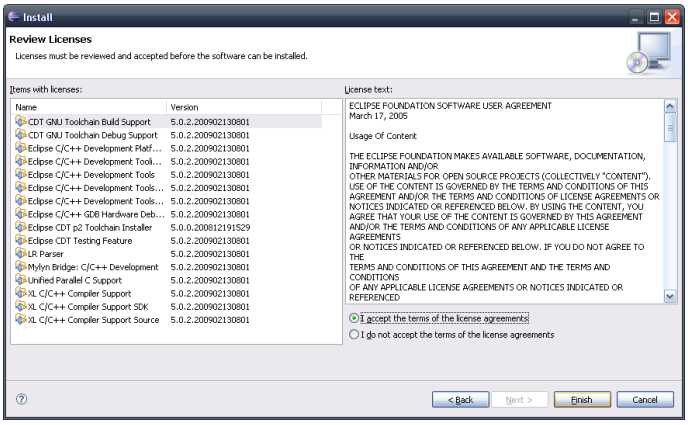


Figura 104: Instalación CDT 3

Reiniciar Eclipse para aplicar los cambios.

9.3.3. Montado de los proyectos

9.3.3.1. Sistema con interfaz gráfica OpenCV

Abrir Eclipse.

Crear un proyecto para C++.

Copiar a la carpeta de fuentes el contenido de la carpeta src.

Copiar el archivo de configuración a la raíz del proyecto.

Incluir librerías de OpenCV.

Seleccionar el proyecto en el Project Explorer y acceder a la opción de la barra de menú Project -> Properties -> C/C++ Build -> Settings -> Tool Settings

En el apartado GCC C++ Compiler -> Directories, añadir en Include paths:

"C:**<directorio de OpenCV>**\cv\include"

"C:**<directorio de OpenCV>**\cvaux\include"

"C:**<directorio de OpenCV>**\cxcore\include"

"C:**<directorio de OpenCV>**\otherlibs\cvcam\include"

"C:**<directorio de OpenCV>**\otherlibs\highgui"

En el apartado MinGW C++ Linker -> Libraries, añadir en Libraries:

cv

cvaux

cxcore

cvcam

highgui

En el apartado MinGW C++ Linker -> Libraries, añadir en Library search path:

"C:\ **<directorio de OpenCV>**\lib"

Incluir librerías para el envío de información por UDP.

Seleccionar el proyecto en el Project Explorer y acceder a la opción de la barra de menú Project -> Properties -> C/C++ Build -> Settings -> Tool Settings

En el apartado MinGW C++ Linker -> Libraries, añadir en Libraries:

ws2_32

En el apartado MinGW C++ Linker -> Libraries, añadir en Library search path:

"C:\MinGW\lib"

Compilación de los proyectos

Seleccionar el proyecto en el Project Explorer y acceder a la opción de la barra de menú Project -> Build All

Limpieza de los proyectos

Seleccionar el proyecto en el Project Explorer y acceder a la opción de la barra de menú Project -> Clean

9.3.3.2. Sistema con interfaz gráfica Visual Studio

Para abrir el proyecto se puede hacer de dos modos:

- Abrir Visual Studio 2008.
  En el menú File -> Open -> Proyect/Solution seleccionar *Interfaz.sln*.
- Ejecutar *Interfaz.sln* (doble clic)

Después de abrirlo hay que ajustar en Motor las rutas de OpenCv y videoInput haciendo clic con el botón derecho en el proyecto *Motor* -> Properties -> C/C++ -> General -> Aditional Include directories y en Properties -> linker -> General -> Aditional Include directories. Si la librería del OpenCV está en *C:/Archivos de*

programas/OpenCV y la librería del videoInput se a descomprimido en *C:/Archivos de programas/OpenCV/lib* no haría falta hacer nada.

9.3.3.3. Receptor UDP

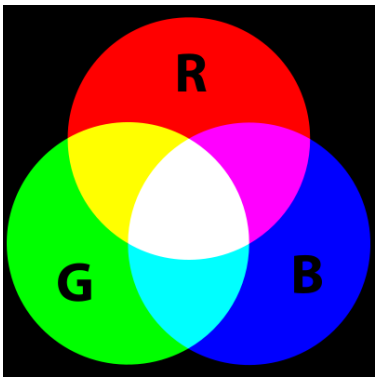
Abrir NetBeans.

En el menú File -> Open Proyect seleccionar el proyecto Receptor

Incluir la librería de dibujado haciendo clic con el botón derecho en el proyecto -> Libraries -> add jar/.. añadir el jar de la librería processing (*core.jar*).

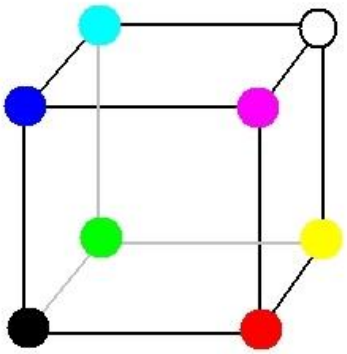
9.4. Apéndice D. Modelos de color

9.4.1. RGB



La descripción RGB (del inglés Red, Green, Blue; "rojo, verde, azul") de un color hace referencia a la composición del color en términos de la intensidad de los colores primarios con que se forma: el rojo, el verde y el azul. Es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores luz primarios. El modelo de color RGB no define por sí mismo lo que significa exactamente rojo, verde o azul, es decir, los mismos valores RGB pueden mostrar colores notablemente diferentes en diferentes dispositivos que usen este modelo de color. Aunque utilicen un mismo modelo de color, sus espacios de color pueden variar considerablemente.

Para indicar con qué proporción mezclamos cada color, se asigna un valor a cada uno de los colores primarios, de manera que, por ejemplo, el valor 0 significa que no interviene en la mezcla y, a medida que ese valor aumenta, se entiende que aporta más intensidad a la mezcla. Aunque el intervalo de valores podría ser cualquiera (valores reales entre 0 y 1, valores enteros entre 0 y 37, etc.), es frecuente que cada color primario se codifique con un byte (8 bits). Así, de manera usual, la intensidad de cada una de las componentes se mide según una escala que va del 0 al 255.



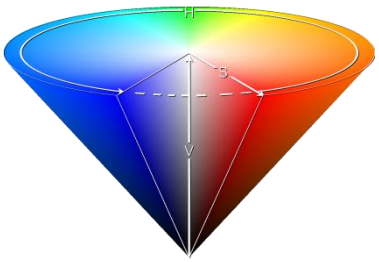
Por lo tanto, el rojo se obtiene con (255, 0, 0), el verde con (0, 255, 0) y el azul con (0, 0, 255), obteniendo, en cada caso un color resultante monocromático. La ausencia de color —lo que se conoce como color negro— se obtiene cuando las tres componentes son 0, (0, 0, 0).

La combinación de dos colores a nivel 255 con un tercero en nivel 0 da lugar a tres colores intermedios. De esta forma el amarillo es (255, 255, 0), el cyan (0, 255, 255) y el magenta (255, 0, 255).

Obviamente, el color blanco se forma con los tres colores primarios a su máximo nivel (255, 255, 255).

El conjunto de todos los colores se puede representar en forma de cubo. Cada color es un punto de la superficie o del interior de éste. La escala de grises estaría situada en la diagonal que une al color blanco con el negro.

9.4.2. HSV



El modelo HSV (del inglés Hue, Saturation, Value – Tonalidad, Saturación, Valor), también llamado HSB (Hue, Saturation, Brightness – Tonalidad, Saturación, Brillo), define un modelo de color en términos de sus componentes constituyentes en coordenadas cilíndricas:

- **Tonalidad**, el tipo de color (como rojo, azul o amarillo). Se representa como un grado de ángulo cuyos valores posibles van de 0 a 360° (aunque para algunas aplicaciones se normalizan del 0 al 100%). Cada valor corresponde a un color. Ejemplos: 0 es rojo, 60 es amarillo y 120 es verde.
- **Saturación**. Se representa como la distancia al eje de brillo negro-blanco. Los valores posibles van del 0 al 100%. A este parámetro también se le suele llamar "pureza" por la analogía con la pureza de excitación y la pureza colorimétrica de la colorimetría. Cuanto menor sea la saturación de un color, mayor tonalidad grisácea habrá y más decolorado estará. Por eso es útil definir la insaturación como la inversa cualitativa de la saturación.
- **Valor del color**, el brillo del color. Representa la altura en el eje blanco-negro. Los valores posibles van del 0 al 100%. 0 siempre es negro. Dependiendo de la saturación, 100 podría ser blanco o un color más o menos saturado.

Se trata de una transformación no lineal del espacio de color RGB, y se puede usar en progresiones de color.

9.4.2.1. Transformación de RGB a HSV

Sea (r, g, b) un color definido en el espacio RGB, perteneciendo esos valores al rango [0, 1], y sea max el valor máximo de los componentes, y min el valor mínimo de esos mismo valores, los componentes del espacio HSV se pueden calcular como:

$$h = \begin{cases} 0 & \text{if } \max = \min \\ (60° \times \frac{g-b}{\max - \min} + 360°) \bmod 360°, & \text{if } \max = r \\ 60° \times \frac{b-r}{\max - \min} + 120°, & \text{if } \max = g \\ 60° \times \frac{r-g}{\max - \min} + 240°, & \text{if } \max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } \max = 0 \\ \frac{\max - \min}{\max} = 1 - \frac{\min}{\max}, & \text{otherwise} \end{cases}$$

$$v = \max$$

9.4.2.2. Transformación de HSV a RGB

Dado un color definido por (h, s, v) en el espacio HSV, con h tomando un valor del intervalo [0, 360), indicando el ángulo en grados de la tonalidad, y con s y v en el rango [0, 1] representando la saturación y el valor del color, respectivamente, el correspondiente color definido por (r, g, b) en el espacio RGB puede ser calculado de la siguiente manera:

$$h_i = \left\lfloor \frac{h}{60} \right\rfloor \mod 6$$

$$f = \frac{h}{60} - \left\lfloor \frac{h}{60} \right\rfloor$$

$$p = v \times (1 - s)$$

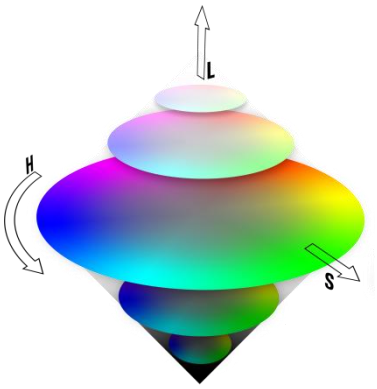
$$q = v \times (1 - f \times s)$$

$$t = v \times (1 - (1 - f) \times s)$$

$$(r, g, b) = \begin{cases} (v, t, p), & \text{if } h_i = 0 \\ (q, v, p), & \text{if } h_i = 1 \\ (p, v, t), & \text{if } h_i = 2 \\ (p, q, v), & \text{if } h_i = 3 \\ (t, p, v), & \text{if } h_i = 4 \\ (v, p, q), & \text{if } h_i = 5 \end{cases}$$

9.4.3. HLS

El modelo HSL (del inglés Hue, Saturation, Lightness – Tonalidad, Saturación, Luminancia), también llamado HSI (del inglés Hue, Saturation, Intensity – Tonalidad, Saturación, Intensidad), define un modelo de color en términos de sus componentes constituyentes. El modelo HSL se representa gráficamente como un cono doble o un doble hexágono. Los dos vértices en el modelo HSL se corresponden con el blanco y el negro, el ángulo se corresponde con la tonalidad, la distancia al eje con la saturación y la distancia al eje



blanco-negro se corresponde a la luminancia. Como el modelo HSV, es una deformación no lineal del espacio de color RGB.

HSL es similar al modelo HSV pero refleja mejor la noción intuitiva de la saturación y la luminancia como dos parámetros independientes:

- En HSL, la componente de la saturación va desde el completamente saturado hasta el gris equivalente, mientras que en HSV, con V al máximo, va desde el color saturado hasta el blanco, lo que no es muy intuitivo.
- La luminancia en HSL siempre va desde el negro hasta el blanco pasando por la tonalidad deseada, mientras que en HSV la componente V se queda a mitad camino, entre el negro y la tonalidad escogida.

9.4.3.1. Transformación de RGB a HLS

Sea (r, g, b) un color definido en el espacio RGB, perteneciendo esos valores al rango [0, 1], y sea max el valor máximo de los componentes, y min el valor mínimo de esos mismo valores, los componentes del espacio HLS se pueden calcular como:

$$h = \begin{cases} 0 & \text{if } \max = \min \\ (60^\circ \times \frac{g-b}{\max - \min} + 360^\circ) \bmod 360^\circ, & \text{if } \max = r \\ 60^\circ \times \frac{b-r}{\max - \min} + 120^\circ, & \text{if } \max = g \\ 60^\circ \times \frac{r-g}{\max - \min} + 240^\circ, & \text{if } \max = b \end{cases}$$

$$l = \tfrac{1}{2}(\max + \min)$$

$$s = \begin{cases} 0 & \text{if } \max = \min \\ \frac{\max - \min}{\max + \min} = \frac{\max - \min}{2l}, & \text{if } l \leq \frac{1}{2} \\ \frac{\max - \min}{2 - (\max + \min)} = \frac{\max - \min}{2 - 2l}, & \text{if } l > \frac{1}{2} \end{cases}$$

9.4.3.2. Transformación de HLS a RGB

Dado un color definido por (h, l, s) en el espacio HLS, con h tomando un valor del intervalo [0, 360), indicando el ángulo en grados de la tonalidad, y con l y s en el rango [0, 1] representando la luminancia y la saturación, respectivamente, el correspondiente color definido por (r, g, b) en el espacio RGB puede ser calculado de la siguiente manera:

$$q = \begin{cases} l \times (1 + s), & \text{if } l < \frac{1}{2} \\ l + s - (l \times s), & \text{if } l \geq \frac{1}{2} \end{cases}$$

$$p = 2 \times l - q$$

$$h_k = \frac{h}{360}$$

$$t_R = h_k + \frac{1}{3}$$

$$t_G = h_k$$

$$t_B = h_k - \frac{1}{3}$$

$$\text{if } t_C < 0 \rightarrow t_C = t_C + 1.0 \quad \text{for each } C \in \{R, G, B\}$$

$$\text{if } t_C > 1 \rightarrow t_C = t_C - 1.0 \quad \text{for each } C \in \{R, G, B\}$$

$$Color_C = \begin{cases} p + ((q-p) \times 6 \times t_C), & \text{if } t_C < \frac{1}{6} \\ q, & \text{if } \frac{1}{6} \leq t_C < \frac{1}{2} \\ p + \left((q-p) \times 6 \times (\frac{2}{3} - t_C)\right), & \text{if } \frac{1}{2} \leq t_C < \frac{2}{3} \\ p, & \text{otherwise} \end{cases}$$

$$\text{for each } C \in \{R, G, B\}$$